TEKNILLINEN KORKEAKOULU

Tietotekniikan osasto

Jari Mäntylä

KOMPONENTTIKUVAUSKANTA OSANA YRITYKSEN JAVA EE -YMPÄRISTÖN HALLINTAA JA SOVELLUSKEHITYSTÄ

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Helsingissä 16.10.2006.

Työn valvoja

professori Antti Ylä-Jääski

Työn ohjaaja

fil. kand. Kari Makkonen

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto	DIPLOMITYÖN TIIVISTELMÄ
Tekijä Jari Mäntylä	Päiväys 3.10.2006
	Sivumäärä 83
Työn nimi Komponenttikuvauskanta osana yrityksen Java EE -ympäristön hallintaa ja sovelluskehitystä	
Professuuri Tietoliikenneohjelmistot	Koodi T-110
Työn valvoja prof. Antti Ylä-Jääski	
Työn ohjaaja fil. kand. Kari Makkonen	

Komponenttien kuvaaminen on yksi yrityksen Java EE -ympäristön hallintaan liittyvistä kolmesta osa-alueesta. Kaksi muuta ovat versionhallinta ja paketointi sekä konfiguraation hallinta. Kaikkia näitä yhdessä kutsutaan sovelluskehityksen hallintavälineiksi. Tutkimuksessa keskityttiin komponenttikuvauskantaan, jota käytetään yrityksen komponenttien kuvaamiseen yhtenäisellä tavalla. Tutkimuksen keskeisin tavoite oli löytää toteutusmalli komponenttikuvauskannalle sekä määritellä komponenteista kuvattavat tiedot. Myös komponenttikuvauskannan liittymät muihin hallintavälineisiin olivat tärkeässä roolissa. Lisäksi tutkimuksessa havainnollistettiin välineiden käytöllä saavutettavia parannuksia kehittämisen tehokkuuteen.

Tutkimuksen alkupuolella määriteltiin komponentteihin ja niiden hallintaan keskeisesti liittyvät käsitteet: komponentti, rajapinta, uudelleenkäyttö, Component-Based Software Delevelopment -sovelluskehitysmalli ja Software Configuration Management -järjestelmät. Käsitteiden jälkeen esiteltiin tarkemmin kunkin sovelluskehityksen hallintavälineen toiminnallisuus liittymineen.

Tutkimuksessa kävi ilmi, ettei valmista toteutusmallia olisi löydettävissä akateemisesta maailmasta tai teollisuudesta. Myöskään kohdeyritykseen soveltuvaa komponenttikuvauskantavälinettä tai sen sisältävää hallintavälinekokonaisuutta ei ole tullut vastaan tutkimuksen aikana. Tästä seurasi, että toteutusmalli muodostettiin kohdeyrityksen sisäisten tarpeiden mukaisesti.

Toteutusmallin kannalta oli oleellista esitellä kohdeyrityksen sovelluskehitysprosessiin osallistuvat roolit ja toimintaympäristö. Näiden jälkeen määriteltiin komponenttikuvauskannan vaatimukset, joihin sisältyvät komponenteista kuvattavat tiedot ja kuvauskannan toiminnallisuus. Seuraavaksi kuvattiin vaatimusten pohjalta toteutettu komponenttikuvauskanta. Kohdeyrityksen toteutus arvioitiin tutkimusteorian pohjalta, sillä toteutusta ei ehditty ottaa käyttöön tutkimuksen aikana.

Tutkimuksen keskeisimmät tulokset olivat komponenttikäsitteen rajaus, konkreettisen toteutusmallin muodostaminen sekä komponenteista kuvattavien tietojen kerääminen ja kokoaminen. Välineiden käytöllä johdettiin saavutettavan parannuksia kehittämisen tehokkuuteen mm. tehokkaamman tiedotuksen, helpomman viestinnän ja nopeamman hallinnoinnin seurauksena. Myös uudelleenkäytettävyyden ja siihen perustuvan sovelluskehitysmallin kriittinen tarkastelu oli tärkeää tulevaisuuden haasteiden tunnistamisen vuoksi.

Avainsanat

Komponenttikuvauskanta, Java EE, Komponentti, Komponentteihin perustuva sovelluskehitys, Konfiguraation hallinta, Paketointi, Versionhallintajärjestelmä, Uudelleenkäyttö, Muutosten vaikutusalueen kartoitus

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author
Jari Mäntylä

Date
October 3, 2006

Pages
83

Title of thesis
The component repository as part of enterprise Java EE software development and management

Professorship
Telecommunications software

Professorship Code
T-110

Supervisor
Antti Ylä-Jääski Dr.Sc.(Tech.)

Instructor
Kari Makkonen B.Sc.

Enterprise Java EE software management comprises three main elements: component description, version control and packeting and configuration management. Together these are called software development administration tools. This study concentrates on the component repository, which is used for describing components in a consistent manner. The main goals of this thesis were to create an implementation model for the repository and define the content of the component description. The importance of integrating the component repository with other administration tools is likewise emphasised. In addition, the thesis points out the improvements attained using these software development tools.

The thesis begins by defining the main concepts related to components and their management: component, interface, software reuse, the Component-Based Software Development paradigm and Software Configuration systems. Each software development administration tool is introduced in detail.

During the research it became obvious that there were no existing implementation models available either in the academic world or the industry. No existing component repository or administration toolkit appropriate for the target corporation was found either. Consequently, the implementation model was created according to the target corporation's internal requirements.

It was essential for the implementation model to present the roles and the operational environment related to the target corporation's software development process. The functional requirements for the component repository and the component description were defined on this basis. The actual implementation model is then described and evaluated based on theory, since the model was not deployed as part of this study.

The main outcomes of the research were a component definition, a concrete implementation model and the gathering and composition of the component description. The thesis confirmed that by using administration tools, one achieves more efficient and easier communication and faster administration. Furthermore, the critical evaluation of software reuse and CBSD was important in identifying future challenges in software development.

Keywords
Component Repository, Java EE, Component, Component-Based Software Development, Software Configuration Management, Packaging, Version Control System, Software Reuse, Impact Analysis, Change Management

Alkusanat

Tämä diplomityö on tehty tammi-lokakuussa 2006 Osuuspankkikeskus Osk:lle osana yrityksen ICT-organisaatiossa läpivietyjä hallintavälineiden ja työtapojen kehittämishankkeita, joilla tavoiteltiin tehokkaampaa sovelluskehittämistä. Yhden hankkeen vastuulla oli juuri luoda komponenttikuvauskanta kehittäjille.

Haluan erityisesti kiittää työn ohjaajaa fil. kand. Kari Makkosta. Hänellä oli selkeä näkemys siitä, miten miten aihealue kannattaa rajata, mitä tutkimuksessa kannattaisi tarkastella ja mikä työssä on olennaista. Karin antama palaute keskeneräisistä versioista oli suureksi avuksi. Haluan myös kiittää työni valvojaa professori Antti Ylä-Jääskiä joustavuudesta diplomityöaiheen valinnassa. Kiitos myös esimiehilleni ja projektipäällikölleni Osuuspankkikeskuksessa ajan järjestämisestä tämän tutkimuksen tekemiselle.

Vanhempiani haluan kiittää kiinnostuksesta opiskelijani kohtaan sekä taloudellisesta tuesta opiskeluaikanani. Opiskeluaikaiset ystäväni ovat tehneet opiskelijan elämästä opiskelijan elämää, kiitos siitä.

Helsinki, 3.10.2006

Jari Mäntylä

Taulukot

Kuvat

SISÄLLYSLUETTELO

TERMIT JA LYHENTEET

API	Application Programming Interface – Ohjelmointirajapinta.
Asentaja	Henkilö, joka vastaa sovellusten ja komponenttien asentamisesta järjestelmätesti- ja tuotantoympäristöihin.
Bean-Managed Transaction (CMT)	Tapahtuma, jonka hallinta hoidetaan manuaalisesti EJB:n toimesta. Katso myös Container-Managed Transaction (CMT).
CBSD / CBD / CBSE	Component-Based (Software) Development/Engineering – Komponentteihin perustuva sovelluskehitysmalli. Sovelluskehityksen kokonaismenetelmä, missä sovellukset koostetaan valmiista komponenteista.
Component Repository	Komponenttikuvauskannan yleisimmin käytetty nimitys englanninkielisissä lähteissä.
Container-Managed Transaction (CMT)	Tapahtuma, jonka hallinta on luovutettu Java EE-palvelinohjelmiston vastuulle. Katso myös Bean-Managed Transaction (CMT).
Doclet API	Rajapinta, joka mahdollistaa laajennuksien toteuttamisen Javadoc-generointiin.
Enterprise Archive (EAR) *Tiedostomuoto*	Java EE-ympäristön sovelluspaketti, joka koostetaan JAR-, EJB-, WAR- ja RAR-komponenteista sekä muista apuresursseista. [http://java.sun.com/blueprints/code/namingconventions.html]
Enterprise JavaBean (EJB)	EJB-komponentit ovat Java EE-palvelinympäristössä toimivia hajautettuja ja etäkutsuttavia palveluja, joiden tapahtumanhallinta on konfiguroitavissa. [http://java.sun.com/products/ejb/]
intranet	Yrityksen sisäinen yksityinen verkko. Tavanomaisesti intranet koostuu useista yhteen liitetyistä lähiverkoista. Lähiverkot voidaan yhdistää myös julkisten verkkojen yli joko vuokraamalla omat tietoliikenneyhteydet tai suojaamalla (ja salaamalla) liikenne ohjelmallisesti.
Internet	Julkinen tietoverkko, joka koostuu lukemattomista lähiverkoista ympäri maailman ja johon pääsee mukaan kytkeytymällä vähintään yhteen verkon reitittimeen.
Java	Ohjelmointikieli ja suoritusympäristö, jonka perusidea on alustariippumaton kehitys. Lähdekoodissa ei tarvitse ottaa huomioon alustan teknisiä yksityiskohtia, koska käännettyä koodia suoritetaan Java-virtuaalikoneessa, joka piilottaa alla olevien järjestelmien eroavaisuudet. [http://java.sun.com/]

Java Archive (JAR) *Tiedostomuoto*	Paketti, joka koostetaan käännetyistä Java-luokista ja muista apuresursseista. JAR on samantapainen arkistotiedostomuoto kuin yleisesti tunnettu ZIP. JAR-paketti eroaa ZIP-paketista mm. sisältämänsä manifest-tiedoston osalta. Manifest-tiedosto kuvaa JAR:n sisältöä ja tarjoaa tarkistussummia. [http://java.sun.com/docs/books/tutorial/jar/, http://mindprod.com/jgloss/jar.html#JARVZIP]
Java ClassLoader	ClassLoader on Java-virtuaalikoneen osa, joka lataa luokat muistiin aina tarvittaessa. ClassLoader on toteutettu Java-kielellä ja siksi sen toimintatapaa on mahdollista muuttaa periyttämällä tästä oma toteutuksensa. Varsinkin monet Java EE -palvelimet (ks. alla) hyödyntävät tätä mahdollisuutta toteutuksessaan.
Javadoc	Työkalu, jonka avulla generoidaan HTML-muotoinen rajapintadokumentaatio (API) Java-lähdekoodiin kirjoitettujen kommenttien perusteella.
Java EE (ennen J2EE)	Java Platform, Enterprise Edition. Sun Microsystemsin kehittämä laaja kokoelma rajapintamäärittelyjä, jotka yhdessä muodostavat perustan samannimiselle monitasoarkkitehtuurille. Tämä tunnettiin aiemmin nimellä Java 2 Platform, Enterprise Edition (J2EE). [http://java.sun.com/javaee/]
Java EE -palvelin	Palvelinohjelmisto (Application Server), joka toteuttaa Java EE -määrittelyn mukaiset rajapinnat ja mahdollistaa Java EE -monitasoarkkitehtuurin mukaisesti toteutettujen sovellusten suorittamisen. Kehittäjä voi halutessaan jättää palvelinohjelmiston vastuulle monimutkaisemmat tehtävät, kuten esim. tapahtumanhallinnan.
Java SE (ennen J2SE)	Java Platform, Standard Edition. Sun Microsystemsin kehittämä Java-ohjelmointikielen perusversio. Java SE sisältää monia toiminnallisuuksia useilla perusalueilla, kuten esim. tietokantayhteydet, tietoturva, etäkutsut (Remote Method Invocation – RMI) ja tietoliikenne. Java EE sisältää useita Java SE:n toiminnallisuuksia laajentavia teknologioita. [http://java.sun.com/javase/]
JavaServer Pages (JSP)	JSP-tiedostot ovat Java EE-palvelinympäristössä toimivia web-sivuja, jotka sisältävät XML-tyyppisiä lausekkeita ja Java-kieltä HTML-koodin sekaan upotettuna. [http://java.sun.com/products/jsp/]
JavaServer Pages (JSP) Tag Libraries	Tag Libraries -toteutukset mahdollistavat yhtenäisen ja modulaarisen tavan käsitellä tiiviissä muodossa tietoja JSP-sivulla. Päätarkoituksena on yhtenäistää tiedonesitystapoja sekä vähentää JSP-sivulla käytettävän Java-kielen määrää.

JDK	Java Development Kit. JRE:n laajennus, joka sisältää lisäksi työkalut Java-lähdekoodien kääntämiseen (compilers) ja virheiden etsimiseen (debuggers).
JRE	Java Runtime Environment. Sisältää Java-virtuaalikoneen ja tarvittavat osat Java-kielellä kirjoitettujen ohjelmien suorittamiseen.
Kehittäjä	Henkilö, joka vastaa sovellusten/komponenttien teknisestä suunnittelusta sekä toteutuksesta.
Paketoija	Henkilö, joka vastaa sovellusten/komponenttien kääntämisestä ja koostamisesta.
Resource Adapter (RAR) *Tiedostomuoto*	Sovitinkomponenttipaketti, jonka avulla tietty EIS-järjestelmä (Enterprise Information System) integroidaan yhtenäisellä tavalla Java EE-palvelimeen.
SCM	Software Configuration Management – Sovellusjärjestelmien konfiguraatioiden hallinta -paradigma. SCM käsittää sovelluskehitysprosessissa luotujen artifaktien ja versioiden hallinnoimisen sekä järjestelmien kehityksen seurannan.
Tapahtumanhallinta	Hallintatoimet, jotka liittyvät yhden tapahtumakokonaisuuden eheään suorittamiseen. Tarkoituksena on taata, että joko koko tapahtumaketju suoritetaan onnistuneesti tai että yhtäkään tapahtumaketjun tehtävää ei suoriteta.
Toimitusseloste	Dokumentti, johon kirjataan ohjeet ja avaintiedot sovellusten ja komponenttien siirtämiseksi ja asentamiseksi järjestelmätesti- ja tuotantoympäristöihin. Dokumentin täyttävät kehittäjät ja paketoija. Dokumenttia hyödyntävät lopulta asentajat.
Web Application Archive (WAR) *Tiedostomuoto*	Verkkosovelluspaketti, joka koostetaan käännetyistä Java-luokista, JSP-sivuista, staattisista webresursseista ja muista apuresursseista. [http://java.sun.com/Java EE/tutorial/1_3-fcs/doc/WebComponents3.html]

1. JOHDANTO

> *"In the world of components, components are not easy to create properly, and applications are not easy to create, even when using well designed components."*
>
> *Rebecca Parsons, Components and the World of Chaos [Par03]*

Uudelleenkäytettävien komponenttien kuvausta ja hallinnointia on tutkittu jo useiden vuosien ajan [Boo87, Pri91, Bor95, Beh98, SHW98, Gri99]. Lähes kaikki aihealuetta käsittelevät kirjoitukset pohtivat uudelleenkäytettävyyttä pääasiallisesti teknisistä, inhimillisistä ja kognitiivisista näkökulmista. On kuitenkin kiinnitetty huomattavasti vähemmän huomiota toimivien komponenttikuvauskantojen rakentamiseen, asentamiseen ja ylläpitämiseen muuttuvien tarpeiden maailmassa [Hen96]. Myös Vitharana [Vit03] on havainnut, että komponenttien luokittelua, hakua ja jäljittämistä on tutkittu hyvin rajallisesti. Uudelleenkäytettävyys ja kehittämisen tehostuminen ovat pääasialliset motivaatiotekijät komponenttien kuvaamiseen ja kirjastoimiseen. Tässä työssä ei kuitenkaan käsitellä mainittavasti komponenttien uudelleenkäytettävyyttä vaan keskitytään sovelluskehitysprosessissa syntyneiden komponenttien hallinnointiin. Komponenttien hallinnointiin sisältyvät seuraavat osakokonaisuudet:

1. versionhallinta ja paketointi
2. komponenttikuvauskanta
3. konfiguraation hallinta

Yhdessä näitä voidaan kutsua sovelluskehityksen hallintavälineiksi. Välineet eivät ole irrallisia työkaluja, vaan niillä on erilaisia riippuvuuksia toisiinsa.

Versionhallinnalla ja paketoinnilla tarkoitetaan lähdekoodin keskitettyä tallentamista, versioimista, kääntämistä ja käännösten paketoimista loogisiksi kokonaisuuksiksi – komponenteiksi. Komponenttikuvauskanta on keskitetty paikka, jonne tallennetaan monipuolisia metatietoja yrityksen komponenteista. Konfiguraation hallinta -välineellä on mahdollista tarkastella ja hallinnoida komponentteja yrityksen monissa suoritusympäristöissä.

Näiden välineiden yhteiskäytöllä tavoitellaan yhtenäistä sovelluskehittämistapaa ja ympäristön hallinnointia. Erityisesti välineillä tavoitellaan lisää tehokkuutta sovelluskehittämiseen. Välineet ovat myös eräänlainen sopimus siitä miten sovelluksia ja palveluita yrityksessä kehitetään. Lisäksi välineet tehostavat eri osapuolien välistä viestintää ja viestinnän sisältöä. Viestinnässä voidaan hyödyntää välineissä käytettyä käsitteistöä ja terminologiaa yhtenäisesti.

1.1 Tutkimuksen taustat

Tämän diplomityön kohdeyrityksen ICT-organisaatiossa tehtiin vuonna 2005 kehitysympäristön välineistön ja työtapojen arviointi sekä kehitettävien asioiden priorisointi haastattelemalla kehittäjiä ja muita kehitysprosessiin osallistuvia osapuolia. Näin saatiin kartoitettua tärkeimmät kehitystarpeet, joiden pohjalta käynnistettiin useita osaprojekteja. Näissä projekteissa sekä parannellaan vanhoja että kehitellään uusia hallintavälineitä ja toimintatapoja. Projektien toimesta tehtävillä muutoksilla tähdätään tehokkaampaan sovellusten kehittämiseen aina niiden määrittelyvaiheista tuotantoon asti.

Kohdeyrityksen komponenttikuvauskanta kehittäjille -projektin tarkoituksena on kartoittaa kaikki yrityksessä olevat komponentit, määritellä kuvauskannan vaatimukset sekä toteuttaa komponenttikuvauskantatuote. Osana projektia tehdään tämä tutkimus komponenttikuvauskannasta ja sen suhteesta yrityksen muihin sovelluskehityksen hallintavälineisiin sekä sovelluskehitysprosessiin. Tutkimuksessa tulisi selvittää myös yritysmaailman ulkopuolella kehiteltyjä ideoita ja toimintatapoja. Tutkimuksen käsittelemältä aihealueelta ei löydy mitään varsinaisia standardeja tai edes vakiintuneita käytäntöjä. Uudelleenkäyttöä on käsitelty sekä teollisuudessa että akateemisessa maailmassa erittäin laajasti koko sovelluskehityksen historian ajan, mutta varsinaisesti komponenttikuvauskannan sisältöön tai toteutusmalliin ei ole useinkaan otettu kantaa. Komponenttikuvauskanta, tai pelkkä komponenttivarasto, onkin monesti mainittu vain sivulauseessa tai sitä on muuten käsitelty vähäsanaisesti. Edellä mainitut seikat osaltaan lisäsivät mielenkiintoa aihealuetta kohtaan ja samalla kasvattivat työn haastavuutta.

1.2 Tutkimuksen tavoitteet

Tutkimuksen keskeisin tavoite on löytää toteutusmalli komponenttikuvauskannalle sekä määritellä komponenttikuvauskannan vaatimukset ja komponenteista kuvattavat tiedot. Vaatimuksista tulee myös käydä ilmi mitä liittymiä komponenttikuvauskanta hyödyntää. Toteutusmallin tulee soveltua nimenomaan kohdeyrityksen sovelluskehitysprosessiin.

Tutkimuksessa on myös tarkoitus tutkia mitkä asiat hallintavälineiden osalta vaikuttavat sovelluskehittämisen tehokkuuteen. Tehokkuuden tarkastelu voidaan keskittää kolmeen kohteeseen: uudelleenkäytön edistäminen, hallinnoinnin tehostuminen ja viestinnän kehittyminen.

1.3 Rajaukset

Muiden hallintavälineiden osalta käsitellään ainoastaan asiaankuuluvat vaatimukset, toiminnallisuus sekä niiden tarjoamat liittymät ja palvelut komponenttikuvauskannalle.

Komponenttikuvauskannassa ja muissa järjestelmissä kuvattavat ja käsiteltävät komponentit ja lähdekoodi rajataan Java-ohjelmointikieleen. Monet käsitellyistä asioista todennäköisesti soveltuisivat käytettäväksi myös muilla ohjelmointikielillä toteutettujen komponenttien yhteydessä, mutta tämän rajauksen tarkoituksena on mahdollisuus keskittyä Java-teknologioihin liittyviin yksityiskohtiin. Itse hallintavälineet voivat olla toteutettu millä teknologialla ja ohjelmointikielellä tahansa.

Vaikka työssä kuvataan kohdeyrityksen sovelluskehitysprosessiin liittyvät roolit ja toimintaympäristö, ei samalla kuitenkaan luoda sovelluskehitysprosessin ohjeistusta tai menetelmiä, sillä ne eivät sellaisenaan liity tutkimuskohteeseen ja työn tavoitteisiin.

Työssä ei oteta kantaa tietoturvallisuuteen, vaikka välineiden käytölle olisi mahdollista asettaa rajoitteita esim. sovelluskehitysprosessiin liittyvien roolien kautta. Yritys, jolle työ tehdään, ei myöskään halua julki tietoturvaan liittyviä seikkoja.

Työssä pohditaan komponenttien uudelleenkäytettävyyttä vain hyvin vähän, vaikka yksi komponenttikuvauskannan tärkeimmistä tehtävistä on tukea uudelleenkäyttöä. Uudelleenkäytettävyys on kuitenkin jo itsessään niin suuri aihealue, että siitä riittää asioita useammankin tutkimuksen aiheeksi.

1.4 Tutkimusmenetelmät

Toteutusmallin löytämiseksi hyödynnetään kohdeyrityksen työntekijöiden kokemuksia sekä aihetta käsittelevää alan kirjallisuutta, tutkimuksia ja artikkeleita. Avoimen lähdekoodin yhteisöstä löytyvät vakiintuneet menetelmät olisivat myös hyvä tarkastelukohde, mutta niistä on löydettävissä hyvin vähän lähdeviitteitä.

Diplomityön kohdeyrityksessä on käynnissä Java-sovelluskehityksen hallintavälineiden osalta projekteja ja hankkeita. Kohdeyrityksen sisäistä tietämystä ei voida kuitenkaan hyödyntää sellaisenaan, vaan asiat on yleistettävä.

1.5 Työn rakenne

Johdannon jälkeen esitellään ja määritellään kaikki tutkimukseen keskeisesti liittyvät käsitteet. Käsitteiden jälkeen esitellään tarkemmin kukin sovelluskehityksen hallintavälineistä. Seuraavaksi esitellään edeltävän teoriapohjan perusteella muodostettu komponenttikuvauskannan toteutusmalli. Oman toteutusmallin johdosta seuraavaksi kuvataan kohdeyrityksen sovelluskehitysprosessiin liittyvät roolit sekä toimintaympäristö. Valitun toteutusmallin ja kohdeyrityksen sovelluskehitysprosessin pohjalta määritellä komponenttikuvauskannan vaatimukset ja kuvataan varsinainen toteutus kohdeyrityksessä. Toteutus myös arvioidaan teoreettiselta pohjalta. Lopuksi esitetään yhteenvetona tutkimuksen johtopäätökset sekä jatkotutkimusmahdollisuudet.

2. KESKEISTEN KÄSITTEIDEN MÄÄRITTELY

2.1 Komponentti

> *"A software component is a physical packaging of executable software with a welldefined and published interface."*
>
> *Jon Hopkins, Component primer [Hop00]*

2.1.1 Määritelmien ylitarjonta

Komponentti-käsitteelle löytyy monta määritelmää, kuten usein informaatioteknologia-alan trendien kohdalla ilmenee [Par03]. Useissa lähteissä käsitellään komponentteihin perustuvaa sovelluskehitysmallia (Component-Based Software Development), minkä yhteydessä kaikki komponentit mielletään uudelleenkäytettäviksi. Komponentti-käsitettä ei ole aina rajattu yksiselitteisesti vaan monesti määritelmä sallii mm. sisäkkäisyyttä, jolloin esim. komponenteista koostettu kompositio on samalla itsekin komponentti. Crnkovic et al. [Crn02] yhdistää oliomaailman ja komponenttikäsitteen seuraavasti: Komponentti on kokoelma olioita, jotka tekevät yhteistyötä ja ovat tiukasti toisiinsa kietoutuneita.

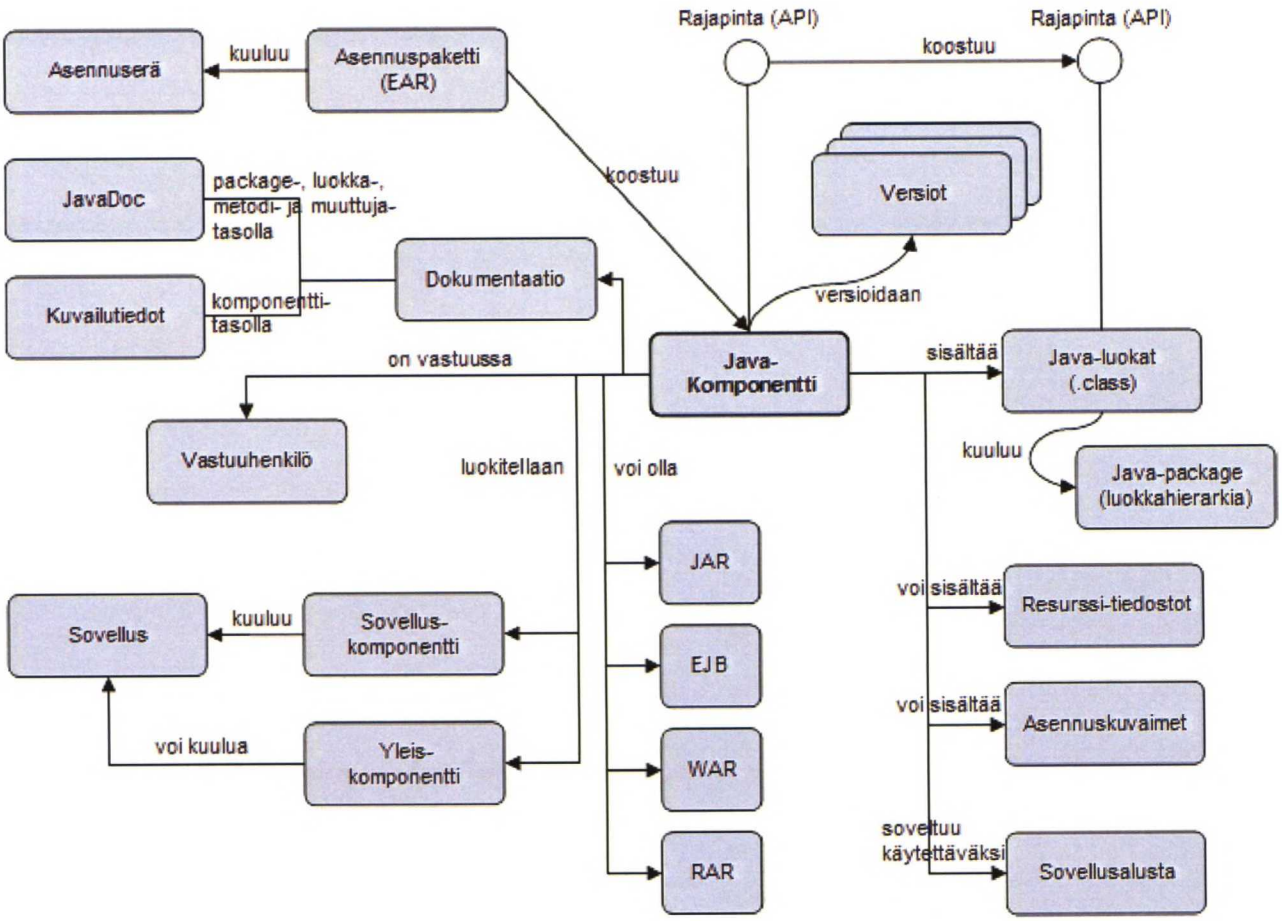
2.1.2 Määritelmän rajaus

Tässä työssä komponentilla tarkoitetaan Java Archive (JAR) -pakettia, jollaiseksi käännökset Java-kielen yhteydessä kapseloidaan. Komponenteiksi lasketaan myös Enterprise JavaBean (EJB) eli palvelukomponentit, Web Archive:t (WAR) eli selainsovellus- ja Web Service -paketit sekä Resource Adapter:t (RAR) eli resurssisovitinpaketit. Komponentti-käsitettä ei mitenkään sidota uudelleenkäytettäviin komponentteihin, vaan komponenteiksi lasketaan kaikki JAR-, EJB-, WAR- ja RAR-paketit. Tämä määrittely on Crnkovic:n [Crn02] oliomaailmaan sijoittuvan komponenttimääritelmän mukainen. Koska uudelleenkäyttö on rajattu ulos komponenttimääritelmästä, tutkimuksessa painotetaan sovelluskehittämisen tehostumista komponenttien parantuneen hallinnan seurauksena.

JAR-pakettiin voi sisältyä erilaisia kuvaimia ja muita resurssitiedostoja, jotka muuttavat komponentin käyttötarkoitusta. Enterprise JavaBean (EJB)-komponentit ovat yksi kuvaimia sisältävä JAR-pakettityyppi. Crnkovic:n komponenttimääritelmää onkin syytä laajentaa ottamalla yhteen kietoutuneiden olioiden (=Java-luokkien) lisäksi määrittelyyn mukaan pakettiin liittyvät resurssitiedostot ja asennuskuvaimet. Nämä ovat osa komponentin rajapintaa, koska niiden avulla on yleensä mahdollista hallita esim. komponentin asennuksen ja suorituksen aikaisia ominaisuuksia tai konfiguroida komponentin sisäistä toiminnallisuutta. Nämä tiedostot voidaan sijoitella joko paketin sisälle tai sen ulkopuolelle. Ulkopuolelle sijoittaminen on joskus järkevää, jotta komponentin konfiguraation muuttaminen olisi mahdollista ilman, että itse komponenttin pakettiin tarvitsisi tehdä muutoksia. Kaikki nämä resurssitiedostot ja asennuskuvaimet ovat olennainen osa komponenttia, koska ilman näitä se saattaa olla käyttökelvoton.

Aasennuspaketteja eli Enterprise Archive:ja (EAR) ei lasketa komponenteiksi. Asennuspaketit koostetaan komponenteista eli JAR-, EJB-, WAR- ja RAR-paketeista sekä asennuskuvaimista. Asennuspaketeilla on keskeinen osa Java-ympäristön hallinnassa sekä palveluiden ja sovellusten asentamisessa, mutta niitä ei kuvata komponenttikuvauskannassa. Asennuspaketteja on kuitenkin mahdollista luoda ja hallinnoida paketointi ja konfiguraation hallinta -välineillä.

Java-komponentiin liittyviä käsitteitä on hahmoteltu alla olevassa kuvassa [Kuva 1] kohdeyrityksen ympäristöä soveltaen.



Kuva 1 – Java-komponentiin liittyvien käsitteiden malli kohdeyrityksen ympäristöä soveltaen

2.1.3 Komponenttien luokittelu

Komponentit voidaan luokitella yksinkertaisesti kahteen alatyyppiin:

- ✓ Sovelluskomponentti
- ✓ Yleiskomponentti

Sovelluskomponentilla tarkoitetaan itse tehtyjä tai ulkopuoliselta toimittajalta tilattuja paketteja, jotka on yleensä tehty ainoastaan yhden sovelluksen käyttöön. Yleiskomponentti sitä vastoin on mikä tahansa komponentti, jota on mahdollista hyödyntää useissa sovelluksissa. Sovelluskomponenttien hyödyntäminen toisessa sovelluksessa vaatii lähes aina muutoksia komponentin lähdekoodiin, jolloin käytännössä syntyy uusi sovelluskomponentti.

Sovellusta voidaan pitää joko yhtenäisenä liiketoiminta- tai sovittuna ylläpitokokonaisuutena. Sovelluksena voidaan pitää myös isomman

sovelluskokonaisuuden tai -palvelun osasovellusta. Hallintavälineillä ei siis hallinnoida sovelluksia.

Yleiskomponenteille voidaan tunnistaa useita eri alatyyppejä, joita yrityksen kuitenkin harvoin kannattaa ottaa luokitteluun mukaan. Sillä tuskin saavutetaan mainittavaa hyötyä komponenttien paikantamisen suhteen kuvauskannasta, mikä on pääasiallinen syy komponenttien luokitteluun. Alatyyppejä ovat mm. yleispalvelu-, integraatio-, infra- (mm. tietoturva, tietokanta-ajurit ja resurssisovittimet), sovelluskehys- ja tuotekomponentit (usein kolmannen osapuolen komponentteja, ks. kappale 2.1.4).

Jos halutaan käyttää tarkempaa kuin kahteen tyyppikategoriaan jakoa, niin edellä mainituista ottaisin esille lähinnä kolme tyyppiä: sovellus-, yleispalvelu- ja integraatiokomponentit. Yleispalvelu toteuttaa jonkin usein tarvitun liiketoiminnallisen palvelun yhteisesti sovitun rajapinnan avulla. Integraatiokomponentit taas toteuttavat teknologian uudelleenkäytettävyyttä. Tämä jako on perusteltavissa sovellus- ja integraatiokehittämisen erilaisuudella. Sovellus- ja yleispalvelukehittäminen edellyttävät liiketoimintaosaamista siinä missä integraatiokehittäminen vaatii teknologiaosaamista. Erityisesti integraatioarkkitehtuuriin nojaavan tai siihen tähtäävän yrityksen on syytä harkita integraatiokomponentti- luokittelutyypin mukaan ottamista, sillä se tukee yrityksen integraatioratkaisujen arkkitehtuurilinjausten huomioimista.

2.1.4 Kolmannen osapuolen komponentit

Kirjallisuudessa on useita erilaisia nimityksiä yrityksen ulkopuolelta hankituille komponenteille. Yacoub et al. [Yac99] luokittelee esimerkinomaisesti kolme vaihtoehtoista komponentin ulkopuolista lähdettä: Commercial-Off-the-Shelf (COTS), Military-Off-the-Shelf (MOTS) ja Government-Off-the-Shelf (GOTS). Yksi kirjallisuudessa ajoittain vastaan tuleva nimitys näille komponenteille on NIH eli Not-Invented-Here-komponentti, mikä terminä korostaa kolmannen osapuolen komponenttien uudelleenkäytön psykologisia haasteita.

Tässä työssä kolmannen osapuolen komponenteiksi lasketaan kaikki maksulliset ja maksuttomat komponentit, jotka on tuotu yritykseen ulkopuolelta ja joiden

lähdekoodit eivät ole yrityksen omistuksessa tai hallinnassa. Nykypäivänä nämä ovat hyvin usein avoimen lähdekoodin yhteisöjen [OSLic] tuottamia komponentteja, joista on monesti tullut alan standardi omassa erityistehtävässään. Tätä kehitystä on kiihdyttänyt avoimen lähdekoodin komponenttien korkeaksi koettu laatu sekä halpa hinta.

Joistakin kolmannen osapuolen komponenteista on lähdekoodit saatavilla, mikä antaa yritykselle mahdollisuuden räätälöidä ko. komponentteja omiin tarpeisiinsa. Tämä tulee tehdä kuitenkin harkiten, sillä samalla todennäköisesti muutetaan ko. komponentin toiminnallisuutta, mikä saattaa aiheuttaa hämmennystä ja virhetilanteita, jos yritys hankkii ulkopuolista osaamista toteutustehtäviinsä eikä komponentteihin tehtyjä muutoksia ole dokumentoitu hyvin. Kolmannen osapuolen komponentit on yleensä suunniteltu juuri uudelleenkäytettäviksi sellaisinaan.

2.2 Rajapinta

> *"The most important feature of a component is the separation of its interfaces from its implementation."*
>
> *Ivica Crnkovic et al., Specification, implementation, and deployment of components [Crn02]*

2.2.1 Komponentin rajapinnan ominaisuudet

Gill [Gil04] on jakanut komponentin rajapinnan neljään osa-alueeseen: puumerkki, rajoitteet, paketointitapa ja ei-toiminnalliset ominaisuudet (signature, constraints, packaging ja non-functional properties). Gill:n mukaan puumerkki määrittelee komponentin toiminnallisuuden. Puumerkki käsittää komponentin ominaisuudet (properties), toiminnot (operations) ja tapahtumat (events). Rajoitteet koskevat mm. rajapinnan toimintoja, joiden esi- ja jälkiehdot (pre- ja post-conditions) ovat yleensä jollain tavalla rajoitettuja (tietotyypit, sallitut arvovälit jne.). Paketointitapa kuvaa komponentin roolin eli miten komponentti suhtautuu muihin komponentteihin ja miten se otetaan osaksi tietyn tyyppistä kontekstia tai käyttöskenaariota. Ei-toiminnallisiksi rajapinnan ominaisuuksiksi Gill laskee mm. tietoturvan, suorituskyvyn ja luotettavuuden.

Crnkovic et al. [Crn02] taas jakaa komponentin rajapinnan toiminnallisiin ja ulkotoiminnallisiin ominaisuuksiin (functional ja extrafunctional properties). Lisäksi he liittävät rajapinnan käsitteeseen sopimukset (contracts). Toiminnalliset ominaisuudet he jakavat vienti- ja tuontirajapinnoiksi eli komponentin ympäristölleen tarjoamiin palveluihin ja ympäristöltään odottamiin palveluihin. Ulkotoiminnalliset ominaisuudet vastaavat pitkälti Gill:n ei-toiminnallisia ominaisuuksia, mutta lisäävät mukaan myös muita ominaisuuksia kuten latenssia (latency), saatavuutta (availability) ja sitkeyttä (robustness). Crnkovic:n sopimukset taas muistuttavat paljon Gill:n rajoitteita. Sopimus käsittää koko komponenttia koskevat rajoitteet (the invariant) sekä kunkin toiminnon esi- ja jälkiehdot.

2.2.2 Java-komponentin rajapinta

Java-komponentti koostuu pääasiallisesti Java-luokista, joilla kullakin on oma rajapintansa. Nämä luokat ja niiden rajapinnat yhdessä muodostavat komponentin rajapinnan. Luokan rajapintaan kuuluvat kaikki ulospäin näkyvät metodit (parametreineen) ja luokkamuuttujat. Java-kielessä näkyvyys määritellään erikseen kunkin metodin ja luokkamuuttujan yhteydessä. Java-kielestä löytyy myös oma rajapinta-määritys, joka voidaan esitellä omassa rajapintaluokkassaan, jonka toteutusluokat kertovat toteuttavansa. Tämä mahdollistaa taustalla olevan implementaation vaihtamisen ilman, että rajapinta muuttuu.

Luokkarajapintojen lisäksi tämän työn määritelmän mukaisen Java-komponentin (ks. kappale 2.1.2) rajapintaan lasketaan mukaan vähintään asennuskuvaimet ja muut komponentin konfiguraation määrittävät tiedostot. Nämä sijoittuvat parhaiten Gill:n [Gil04] määritelmän mukaisen rajapinnan paketointitapaosa-alueelle.

2.2.3 Rajapinnan suunnittelu

Usein komponentin toteutus on syytä aloittaa rajapinnan suunnittelusta. On pohdittava mitä toiminnallisuuksia komponentin vastuulla on ja millaisen rajapinnan avulla ne voidaan toteuttaa. Huonompi vaihtoehto on ensin toteuttaa komponentin vastuulla olevat asiat (Ad Hoc) ja yrittää myöhemmin räätälöidä rajapinta käytettävyydeltään siedettäväksi. Komponenttienkin tapauksessa pätee vanha viisas sanonta: "Hyvin suunniteltu ja on jo puoliksi tehty".

2.2.4 Rajapinnan kestävyys

Kertaalleen määritellyn ja toteutetun rajapinnan muuttaminen myöhemmin ei ole suoraviivaista, koska se vaatii usein tavanomaisen regressiotestauksen lisäksi myös muutoksia rajapinnan palveluja hyödyntävien komponenttien lähdekoodiin. Rajapinta onkin syytä suunnitella todella hyvin ja myöhemmin kiinnittää, kun se on ehditty ottaa käyttöön useammassa kuin yhdessä komponentissa. Jos rajapinnassa havaitaan niin vakavia puutteita tai virheitä, että rajapinnan muuttamisen todetaan olevan välttämätöntä, olisi syytä harkita kokonaan uuden komponentin luomista vanhan rinnalle. Tällä tavalla uuden rajapinnan käyttöönotto voidaan tehdä vaiheistetusti sen sijaan, että muutettaisiin kerralla kaikkia olemassa olevan komponentin korjattua rajapintaa hyödyntäviä komponentteja.

Hyvin suunniteltu rajapinta tehostaa sovelluskehittämistä vähintään kahdella tavalla. Yritys säästää myöhemmin kustannuksissa, joita tulee komponenttien ylläpidosta, sillä hyvin suunniteltuja rajapintoja on harvemmin tarve muuttaa. Hyvin suunniteltua rajapintaa on myös todennäköisesti helpompi hyödyntää muiden toimesta, mikä vähentää toteuttamiseen kuluvaa aikaa.

2.2.5 Rajapinnan dokumentointi (API documentation)

Rajapinnan dokumentointi on syytä tehdä vähintään yhtä huolellisesti kuin luokkien sisäisen toteutuksen kommentointi. Rajapinnan tarjoamia palveluita hyödyntävien osapuolien ei pitäisi olla tarpeen tutkia käyttämänsä komponentin lähdekoodia osatakseen hyödyntää sen tarjoamia palveluita täysimittaisesti. Rajapinnan dokumentoinnissa on huomioitava myös poikkeustilanteet ja erilaiset rajoitteet.

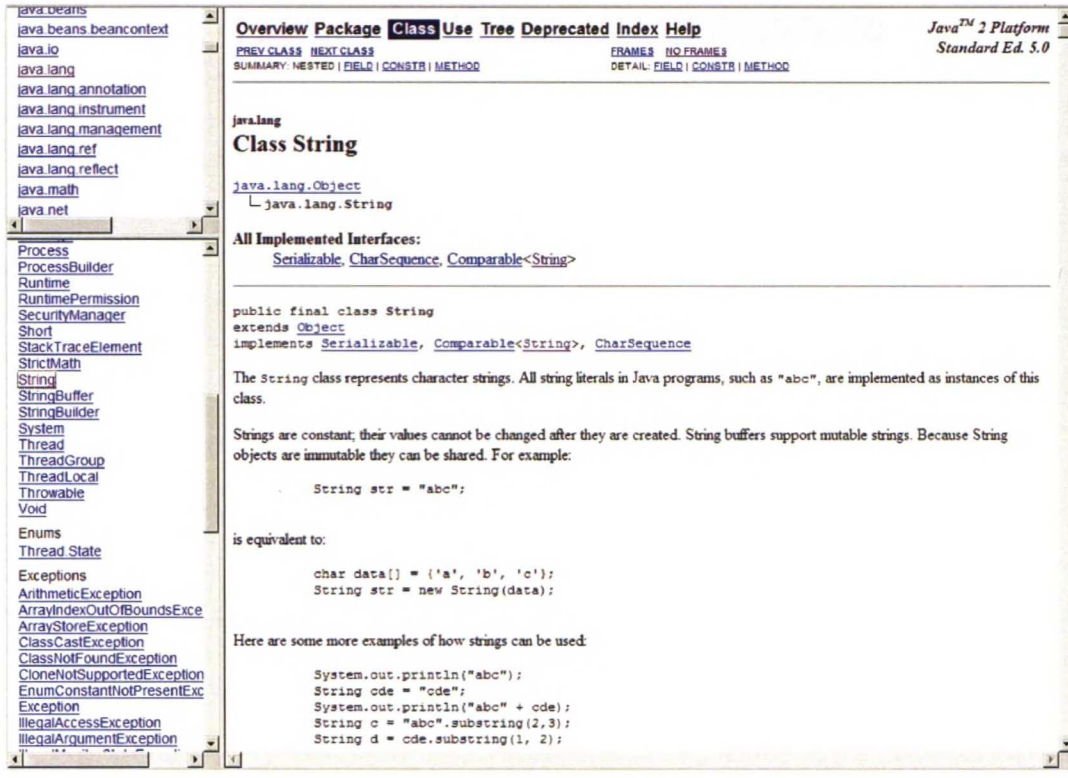
Rajapintadokumentaatiosta pitää selvitä kutsussa käytettävien palveluiden rajapinnassa olevien elementtien (metodien, parametrien ja muuttujien) nimet ja tietotyypit. Parametrien ja muuttujien tapauksessa on erityisesti tarpeen kuvata niitä koskevat rajoitteet. Esimerkiksi jos kysymyksessä on merkkijono (String) tai kokoelma (Collection) -parametri tai -muuttuja, on muotoilu ja rajoitteet syytä kuvata tarkasti. Tämä on parasta ottaa jo rajapinnan suunnitteluvaiheessa huomioon eli on parempi välttää kokonaan tyypittämättömien parametrien tai muuttujien käyttämistä.

Mm. tyypittämättömien kokoelmien (esim. *Map, *List) sijasta tulisi käyttää tyypitettyjä taulukoita (esim. Tilausrivi[]).

Rajapinnaksi lasketaan myös komponenttien välillä liikuteltavat XML-tietueet. Näiden dokumentointi eroaa perinteisimpien kutsurajapintojen dokumentoinnista. Mutta yhtälailla XML-rajapinnan kohdalla on syytä kuvata yksityiskohtaisesti rajapinnassa välitettävien tietojen muotoilu ja rajoitteet.

2.2.6 Javadoc-rajapintadokumentointi

Java-komponenttien tapauksessa rajapinnat on kätevintä dokumentoida lähdekoodiin kirjoitettavien Javadoc-kommenttien avulla. Lähdekoodeista generoidaan Javadoc-työkalun avulla HTML-sivut, joilla näkyy kehittäjän syöttämät kommentit pakettien (package), luokkien, metodien ja muuttujien tasolla. Javadoc-kommentteihin on mahdollista upottaa HTML-elementtejä, joiden avulla kommenttejaan voi muotoilla. Esimerkki Javadoc-dokumentaatiosta on nähtävissä kuvassa alempana [Kuva 2]. Kuvan esimerkissä on hyödynnetty mm. HTML:n <CODE> -tagia, jonka avulla koodiesimerkit ovat luettavampia.



Kuva 2 – Javadoc-esimerkki (J2SE 5.0 java.lang.String)

2.3 Uudelleenkäyttö

2.3.1 Määrittely ja tavoitteet

Uudelleenkäytöllä tarkoitetaan jo kertaalleen toteutetun asian hyödyntämistä myöhemmin uusien sovellusten yhteydessä. Guo ja Luqi [Guo00] kirjoittavat uudelleenkäytön olleen ennen lähinnä opportunistista onnistumista, jossa uusi sovellus on onnistunut hyödyntämään toisen sovelluksen yritelmiä. Heidän mukaansa on erittäin tärkeätä pyrkiä kohti uudelleenkäytön teollistumista, jolloin siitä tulisi erottamaton osa yritysten sovelluskehitysprosessia. Heidän mukaansa uudelleenkäytön tulisi olla systemaattista, jossa hyödynnetään kunkin yrityksen sovellusten ongelma-alueen ja arkkitehtuurin yksityiskohtaisen tarkastelun pohjalta suunniteltuja ja toteutettuja yhteisiä komponentteja.

Kaikkein ollennaisimpina uudelleenkäyttöä edistävinä kehityskohteina he nostavat esiin organisaation infrastruktuurin mukauttamisen asioiden tuotteistamista ja standardointia edistäväksi, komponenttien omistussuhdekriteerien haltuunoton, resurssien investoimisen ja komponenttikuvauskannan perustamisen. Komponenttikuvauskanta on siis vain yksi osatekijä uudelleenkäytön edistämisessä. Pelkkä komponenttikuvauskannan olemassa olo ei riitä, vaan se on myös Guon ja Luqin mukaan oltava voimakkaasti jalkautettu organisaation käyttöön sekä kehittäjälähtöisesti toteutettu.

2.3.2 Uudelleenkäytöllä tavoiteltavat hyödyt

Favaro et al. [Fav98] jakavat uudelleenkäytöllä tavoiteltavat hyödyt operatiivisiin ja strategisiin näkökulmiin. Operatiivisiksi hyödyiksi he luettelevat paremman laadun, paremman tuottavuuden ja alentuneet ylläpitokustannukset. Strategisesti yritykselle saattaa avautua uudelleenkäytön myötä uudet markkinat uudelleenkäytettäväksi tuotettujen komponenttien muodossa. Toisekseen uudelleenkäytön avulla yritys osaa joustavammin vastata kilpailutilanteessa ja muuttuvien markkina-asemien maailmassa. Venäläinen [Ven04] luettelee uudelleenkäyttöön tähtäävän ohjelmistotuotantoprosessin päämääriksi projektin kulujen alentamisen, projektin riskien vähentämisen, projektin elinkaaren lyhentämisen sekä tuotteen laadun

parantamisen. Komponenttien kehittäminen omina kokonaisuuksinaan mahdollistaa myös kunkin komponentin vastuualueen erikoisasiantuntijoiden keskittymisen juuri omaan osaamisalueeseensa, mikä on yksi perustelu uudelleenkäytettävien komponenttien paremmalle laadulle.

2.3.3 Haittaavat tekijät ja ongelmat uudelleenkäytössä

Yksi uudelleenkäytön systemaattisuutta häiritsevistä perustekijöistä on ohjelmistosuunnittelun tapauskohtaisuus ja sitä kautta monimutkaisuus sekä automatisoinnin käytännön mahdottomuus. Järjestelmällinen uudelleenkäyttö ei siis poista tapauskohtaisen suunnittelun tarvetta [Ven04]. Frakes et al. [Fra96] esittävät neljä keskeistä uudelleenkäyttöä vaikeuttavaa ongelmaa. Uudelleenkäyttö ei ole aina yrityksen johdon erityisesti tukema sovelluskehitysprosessin osatekijä vaan kehittäjien itsenäisesti hyödyntämä työtapa. Kehittäjät saattavat myös odottaa johdon edistävän uudelleenkäyttöä esim. palkkioiden muodossa. Uudelleenkäyttö edellyttää myös uudelleenkäytettäviä komponentteja, joiden tuottaminen on kalliimpaa kuin suoraviivaisesti tietyn ongelmajoukon ratkaisevan komponentin tuottaminen. Myös oikeusasiat tuovat omat haasteensa. Mitä oikeuksia komponentin tuottajalla on komponenttiinsa? Millä tavalla ja missä määrin kuluttujalla on oikeus käyttää komponentteja? Oikeusasioiden merkitys on kylläkin hyvin pieni yrityksen sisäisen uudelleenkäytön kohdalla. Uudelleenkäytettävien komponenttien hyödyntäminen vaatii myös erikoistietämystä ja opettelua sovellusten kehittäjiltä. Heidän tulee toimia tiettyjen standardien puitteissa tai muuten heidän tekemisensä eivät istu ollenkaan uudelleenkäyttöä hyödyntävään prosessiin.

Komponenttikuvauskannan perustemisen lisäksi yritysten tulisi erityisesti kiinnittää huomiota Guon ja Luqin [Guo00] mainitsemaan omistussuhteiden muodostamiseen ja komponenttien vastuuttamiseen. Komponenttien uudelleenkäyttö on erittäin vaikeaa, jos niille ei ole nimetty omistajaa tai vastuuhenkilöä, jolta voisi pyytää apua tarvittaessa.

Card et al.:n [Car94] mukaan myös inhimillisillä ja kulttuurisilla tekijöillä on merkittävästi vaikutusta uudelleenkäyttöä edistävien hankkeiden onnistumiseen. Inhimillisiä tekijöitä ovat mm. Not-Invented-Here (NIH) -syndrooma ja yleinen muutosvastarinta. Aiempi näistä merkitsee joidenkin kehittäjien epäluottamusta

muiden tuottamia komponentteja kohtaan, jonka vuoksi he useimmiten toteuttaisivat asian mieluummin itse. Myös kehittäjien koulutuksella on merkitystä. Usein heidät pitää kouluttaa hyödyntämään uudelleenkäytettäviä komponentteja, jolloin heidän intonsa myös uudelleenkäyttöä kohtaan todennäköisesti kasvaa. Tällainen koulutus on kuviteltua vaikeampaa. Se on mm. vaikeampaa kuin jonkin uuden teknologian kouluttaminen. Kehittäjät pitää myös aika ajoin "herättää" uudelleenkäyttöä ajatellen. He saattaavat muuten ohittaa uudelleenkäytön suunnitelmissaan, jos he eivät osaa ja muista aktiivisesti etsiä uudelleenkäytön mahdollisuuksia. Johdon on myös asetettava selkeät mittarit, joilla uudelleenkäytöllä saavutettuja hyötyjä mitataan. Lisäksi johdon on sitouduttava uudelleenkäytön hyötyjen tarkasteluun pitkällä aikavälillä. Lyhytjänteinen johto saattaa pelästyä uudelleenkäyttöprosessin alkupuolen korkeampia kustannuksia ja perääntyä ennakoimattomasti uudelleenkäytön edistämishankkeesta.

2.4 Component-Based Software Development (CBSD)

2.4.1 Määritelmä

CBSD on komponentteihin perustuvaa sovelluskehitystä. Toinen kirjallisuudessa vastaan tuleva nimitys ko. sovelluskehitysparadigmalle on komponenttikeskeinen ohjelmointi (component-oriented programming), joka voidaan nähdä oliokeskeinen ohjelmointi (object-oriented programming) -paradigman johdannaisena.

Brooks [Bro87] ja monet muut ovat jo aikoinaan omaksuneet ideologian "osta, älä toteuta". Määritelmä voisi paremminkin nykypäivänä olla "käytä, älä toteuta". Tämä on komponentteihin perustuvan sovelluskehityksen perusajatus. Sovelluskehitysmallin perusideana on koostaa sovellukset aiemmin toteutetuista tai hankituista komponenteista sen sijaan, että ne toteutettaisiin kokonaisuudessaan itse. Liiketoimintanäkökulmasta mallin päämotivaattorit ovat pienemmät kehityskustannukset, parempi laatu ja helpompi ylläpito. Sovelluskehitysnäkökulmasta taas eduiksi lasketaan esim. standardisointi ja uudelleenkäytettävyys. [Hai97, Tör05]

Komponentteihin perustuva sovelluskehitysmalli siirtää huomion toteutuksesta (implementation) järjestelmien koostamiseen (system composition). Malli sisältää

kaksi selkeästi toisistaan eroavaa perustehtävää: komponenttien toteuttaminen ja hankinta sekä järjestelmien koostaminen. Tässä mallissa yksi komponentin tärkeimmistä ominaisuuksista onkin sen integroitavuus. Sovellusta koostettaessa integroitavuus määrittelee kunkin komponentin kohdalla onko järkevintä hankkia tai toteuttaa uusi komponentti vai uudelleenkäyttää olemassa olevaa [Hai97]. Komponentteihin perustuvan sovelluskehitysmallin yhteydessä onkin erittäin perusteltua ottaa komponenttien kuvailutietoihin mukaan integraatiokomponentti -luokittelutyyppi.

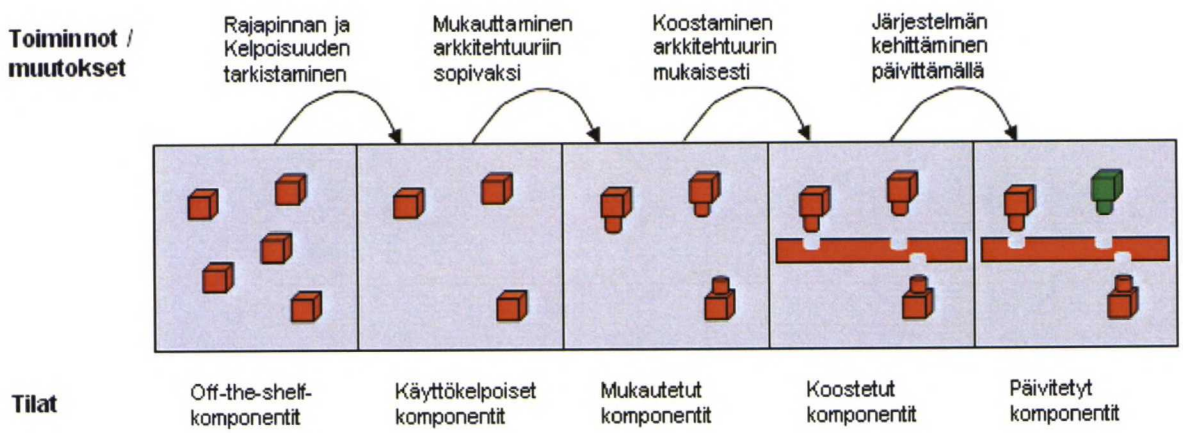
2.4.2 CBSD-Komponentit

Zhang et al. [Zha01] jakaa CBSD:n yhteydessä käytetyt komponentit kahteen luokkaan: primitiivi- (primitive) ja komposiittikomponentteihin (composite component). Ensimmäinen näistä tarkoittaa peruskomponenttia, joka koostuu useista tiedostoista, kuten Java-luokista. Näitä komponentteja syntyy joko alusta asti itse luomalla tai valmiita (primitiivi)komponentteja muokkaamalla. Komposiittikomponentteja taas luodaan yhdistelemällä primitiivikomponentteja ja muita komposiittikomponentteja. Zhang et al. mainitsevat myös välikappaleet (connectors), joita tarvitaan komponenttien yhteen liittämiseksi siten, että komponenttien uudelleenkäytettävyys ei kärsi, kuten kävisi, jos ne aina yritettäisiin liittää tiukasti toisiinsa. Välikappaleet eivät kuitenkaan ole erillisiä rakennuspalikoita vaan yksi komponenttiluokka tai -tyyppi.

Komponentin käsite voi olla CBSD:n yhteydessä hyvin laaja. Clements et al. [Cle95] luettelee erittäin laajan skaalan esimerkkeinä mm. käyttöjärjestelmät, kääntäjät, tietoverkot, tietokantajärjestelmät sekä monet ongelma-aluekohtaiset järjestelmät, kuten esim. lentokoneen navigointialgoritmit. Tämän työn komponenttimääritelmä istuu osaksi CBSD-sovelluskehitysmallia, mutta ei sellaisenaan mitenkään riittäisi komponenttipohjaisen kehittämiseen.

2.4.3 Perustoiminnot

Haines et al. [Hai97] jakavat komponentteihin perustuvan sovelluskehityksen neljään perustoimintoon: komponenttien kartoitus ja kelpoisuuden tarkistaminen, komponenttien sovittaminen, komponenttien kokoonpaneminen eli järjestelmän koostaminen ja järjestelmän kehittäminen. Vaiheet on havainnollistettu kuvassa alla [Kuva 3].



Kuva 3 – Komponentteihin perustuvan sovelluskehityksen perustoiminnot [Hai97]

2.4.3.1 Komponenttien kartoitus ja kelpoisuuden tarkistaminen

Tämä toiminto jakaantuu kahteen alitoimintoon: paikantaminen ja arviointi. Toimintojen tarkoituksena on tutkia olemassa olevien komponenttien soveltumista käytettäväksi osana kulloinkin kehitteillä olevaa järjestelmää. Soveltuvuuden määrittelevät esim. komponentin toiminnallisuus, tarjolla olevat rajapinnat ja mahdollinen standardien käyttö. Valittavana voi olla useita kilpailevia komponentteja. Monesti pitää tarkastella osittain myös komponentin sisäisiä asioita, kuten toteutusalgoritmien validius, testauksen laatu ja määrä, tietoturva jne.

2.4.3.2 Komponenttien sovittaminen

Komponentit on yleensä toteutettu johonkin tiettyyn käyttökontekstiin sopivaksi, mistä seuraa, että ne on usein sovitettava kehitettävän järjestelmän kontekstia. Haines et al. [Hai97] määrittelee kolme erilaista sovittamistapaa: valkoinen, harmaa ja musta laatikko. Valkoinen laatikko -sovittamistapa sallii kattavia muutoksia sovitettavan

komponentin lähdekoodiin. Harmaassa sovittamistavassa lähdekoodiin ei kosketa. Sen sijaan komponentti tarjoaa laajennus- tai ohjelmointirajapinnan (API). Musta laatikko -sovittamisen yhteydessä ei kosketa lähdekoodiin eikä komponentti tarjoa laajennus- tai ohjelmointirajapintaa.

2.4.3.3 Komponenttien kokoonpaneminen eli järjestelmän koostaminen

Koostaminen tehdään käytössä olevan arkkitehtuurin mukaisesti. Kuten Haines et al. [Hai97] arveli jo lähes kymmenen vuotta sitten, on ORB-malli (Object Request Broker) -malli muodostunut näistä kaikkein suosituimmaksi. Tämä arkkitehtuuriteknologia tarjoaa mekanismit ohjelmointikieliriippumattomien rajapintojen määrittelyyn, olioiden paikantamiseen ja aktivointiin.

2.4.3.4 Komponenttien päivittäminen eli järjestelmän kehittäminen

Yksinkertaisimmillaan järjestelmää kehitetään vaihtamalla kehittyneempiä komponentteja aiempien tilalle. Useimmiten tämä on hieman liian optimistinen näkemys, sillä päivitetyt komponentit harvemmin ovat identtisiä aiempien kanssa, minkä vuoksi niille pitää suorittaa huolelliset yksikkö- ja integraatiotestit.

2.4.4 Hyödyt, haasteet, ongelmat ja riskit

Vitharana [Vit03] luottelee komponentteihin perustuvalla sovelluskehitysmallilla tavoiteltavat hyödyt:

- ✓ Nopeammat toimitusajat
  - o Kokonaisten bisnessovellusten koostaminen valmiista komponenteista
- ✓ Alemmat kehityskustannukset
  - o Kertaalleen tehdyn komponentin käyttäminen useassa sovelluksessa
- ✓ Parempi laatu
  - o Komponenttien testaaminen ja käyttö useiden sovellusten yhteydessä
- ✓ Komponentteihin perustuvien sovellusten ylläpito
  - o Korjaavien ja parannettujen komponenttien vaihtaminen tilalle

Samalla hän kuitenkin näkee mallissa useita haasteita, ongelmia ja riskejä, joista on poimittu alle muutamia:

- ✓ Komponenttien ja bisnesongelmien keskinäinen synergia on hankalaa hahmottaa ja komponentin istuminen osaksi bisnesprosessia haastavaa todentaa.
- ✓ Hyvien rajapintojen määrittely ja suunnittelu vaikeaa
- ✓ Komponenttien useiden versioiden hallinnointi, koordinointi ja käyttö useissa eri sovelluksissa
- ✓ Testaaminen monella tasolla (yksikkö-, integraatio-, järjestelmä-, poikkeustesti jne.) jatkuvaa, suuritöistä ja hidasta. Kehittäjien on yksikkötestattava komponentit huolella, sillä he eivät voi tietää miten ja missä konteksteissa niitä käytetään ja keiden toimesta.
- ✓ Komponenttien arviointi on hankalaa ja vaatii omanlaisen metriikkansa. Muutamia erikoisuuksia ovat mm. komponenttien hinnoittelu, käyttölisenssit ja omistussuhteet
- ✓ Komponentteihin perustuva sovelluskehitys vaatii oman ajattelutapansa, johon kehittäjät on koulutettava tai on palkattava uusia osaajia. Osaajia tarvitaan sekä komponenttien kehitys- että koostamistehtäviin.
- ✓ Komponentteihin perustuva sovelluskehitysprosessi vaatii myös huomattavan alkupanostuksen käyttöönoton yhteydessä.
    - o Valinta monien mahdollisten komponenttiteknologioiden välillä erittäin raskasta, sillä välineet ovat kalliita ja tuotosten on oltava linjassa asiakkaiden ja markkinoiden toiveiden kanssa.
    - o Komponentteja jouduttaneen tuottamaan alussa paljon, kun halutun toiminnallisuuden kattavia komponentteja ei löydy valmiina. Kehittäjillä saattaa olla kaksi vaihtoehtoa: tilata mahdollisesti kallis komponentti toteuttamaan haluttu toiminnallisuus tai muuttaa sovelluksen vaatimuksia olemassa olevan komponentin toiminnallisuuksien mukaisiksi.
    - o Yritysten legacy-järjestelmien liittäminen osaksi CBSD-mallia on pakollista ja saattaa osoittautua odotettua kalliimmaksi tai lähes mahdottomaksi.

2.4.5 Arviointi ja soveltaminen

Tässä tutkimuksessa on keskitytty yhteen olioteknologiaan (Java) ja sen tekniikoilla paketoituihin komponentteihin. Crnkovic et al. [Crn02] päättelee, että olioteknologoilla (object-oriented approach) ja komponentteihin perustuvalla

sovelluskehityksellä on selviä yhteneväisyyksiä, mutta myös selviä eroavaisuuksia. CBSD keskittyy enemmän rajapintoihin ja "musta laatikko" -tyyppisten komponenttien uudelleenkäyttöön kun taas olioteknologiat ovat ohjelmointi- ja ohjelmointikielivetoisia. Heidän mukaansa ne eivät kuitenkaan sulje toisiaan pois vaan oliopohjainen analysointi, suunnittelu ja kehitys muodostavat perustan CBSD:lle.

Komponentteihin perustuva sovelluskehitysmalli ei ota juurikaan kantaa siihen miten komponentteja tulisi dokumentoida tai luokitella. Se ei myöskään ota kantaa siihen miten komponentteja varastoidaan tai kirjastoidaan. Tutkimuksessa on kuitenkin haluttu tiedostaa tämän sovelluskehitysmallin pitkä historia ja mahdollisesti lisääntyvä käyttö tulevaisuudessa, mikä saattaa aiheuttaa uudenlaisia muutospaineita ja tavoitteita komponenttikuvauskannalle ja muille sovelluskehityksen hallintavälineille. On todennäköistä, että sovelluskehittäminen lähestyy ajan myötä yhä enemmän komponentteihin perustuvaa mallia.

2.5 Software Configuration Management (SCM) ja CBSD

> *"Configuration management (CM) is a disciplined approach to managing the evolution process of software development and maintenance"*
>
> *Lu Zhang et al., A Configuration Management System Supporting Component-Based Software Development [Zha01]*

2.5.1 Määritelmä

Zhang et al. [Zha01] luettelee SCM-järjestelmien vastuualueelle kuuluvan kehitysprosessissa tuotettujen ohjelmistojen ja komponenttien sekä niiden versioiden hallinnoinnin, niiden kehityksen seuranta ja ylipäätään sovellusten kehittämisessä avustaminen. SCM:n ja CBSD:n yhteisessä mallissa yksi konfiguraatio vastaa yhtä komposiittikomponenttia tai kokonaista järjestelmää.

Zhang et al. [Zha01] mainitsevat myös, että SCM:n ja CBSD:n välistä suhdetta on tutkittu todella vähän. Heidän tutkimuksestaan löytyy hyvä yleistys siitä miten perinteistä hallinnointiprosessia on muunneltava, jotta se soveltuisi komponentteihin perustuvaan sovelluskehitysmalliin.

2.5.2 Tavanomainen sovelluskehitysmalli vs. CBSD-malli

Zhang et al. [Zha01] mainitsee kaksi merkittävintä eroa näiden kahden mallin välillä:

1. Perinteisessä sovelluskehitysmallissa järjestelmä muodostetaan luomalla useita lähdekooditiedostoja, kääntämällä, yhdistämällä ja versioimalla nämä. CBSD-mallissa taas järjestelmä koostetaan yhdistelemällä primitiivi- ja komposiittikomponentteja. Eli SCM:n hallinnoitava peruselementti vaihtuu tiedostosta komponentiksi.
2. Perinteisessä sovelluskehitysmallissa järjestelmä implementoidaan pelkän ohjelmointikielen avulla. CBSD-mallissa ohjelmointikielen avulla luodaan edelleen primitiivikomponentit, minkä lisäksi käytetään komponenttien kuvaus- (component description language) ja arkkitehtuurin kuvauskieliä (architecture description language) komposiittikomponenttien ja järjestelmien koostamista varten. Eli SCM:lle kohdistuu uusia vaatimuksia, kun järjestelmien eheyden tarkistamisessa hyödynnetään useita teknisesti eritasoisia kieliä ja kuvauskeinoja.

2.5.3 SCM:n hallinnointitoimet CBSD-sovelluskehitysmallin osana

SCM:n hallinnointitoimien sovittaminen CBSD-sovelluskehitysmalliin luo teoreettisen mallin komponenttien hallinnoinnille, jonka tutkimuksessa käsitelty sovelluskehityksen hallintavälinekokonaisuus toteuttaa. Nämä hallinnointitoimet muodostavat komponenttien paketointivälineen perustoiminnallisuuden.

CBSD-komponenttien hallinnointiin sisältyvät toiminnot jakaantuvat karkeasti kolmeen luokkaan: primitiivi- ja komposiittikomponentteja koskeviin toimintoihin sekä eheydenhallintatoimintoihin. [Zha01]

2.5.3.1 Primitiivikomponenttien hallinnointi

Primitiivikomponentteja hallinnoidaan kolmen eri perustoiminnon kautta: ulos- ja sisäänkirjaaminen (check-out and check-in), haaroittaminen ja yhdistely (branching and merging) sekä rinnakkaisuuden hallinta (concurrency management). Ulos- ja sisäänkirjaamista käytetään yhden primitiivikomponentin yhden haaran kehittämiseen. Kunkin sisäänkirjaamisen yhteydessä primitiivikomponentista syntyy

uusi versio. Haaroittaminen mahdollistaa uuden haaran luomisen olemassa olevasta primitiivikomponentin versiosta. Myöhemmin haaroja voidaan yhdistellä muodostaen uusia versioita. Rinnakkaisuuden hallintaa tarvitaan, jos sallitaan useamman henkilön samanaikainen työskentely yhden primitiivikomponentin parissa.

2.5.3.2 Komposiittikomponenttien hallinnointi

Komposiittikomponenttien hallinnointiin tarvitaan kaikkiaan neljää toimintoa: konfiguraation luonti (configuration creation), perusviivan luonti (baseline creation), konfiguraation muokkaus (configuration modification) ja konfiguraation haaroittaminen (configuration branching). Konfiguraatioita luodaan yhdistelemällä primitiivikomponentteja ja muita (ali-)konfiguraatiota (sub-configuration). Konfiguraation perusviivan luonnin yhteydessä kunkin sen yhdistämän primitiivikomponentin versio ja alikonfiguraation perusviiva kiinnitetään. Konfiguraatiota voidaan myöhemmin muokata muuttamalla sen koostumusta eli vaihtamalla sen yhdistelemiä primitiivikomponentteja ja alikonfiguraatioita. Primitiivikomponenttien tapaan myös komposiittikomponentteja voidaan haaroittaa.

2.5.3.3 Eheydenhallinta

Eheydenhallinta käsittää kaksi toimintoa: Koostumuksen ja eheyden tarkistaminen (consistency and integrity checking) sekä muutosanalysointi (change affection analysis). Primitiivi- ja komposiittikomponenttien keskinäistä koostumusta ja eheyttä on hyödyllistä seurata – varsinkin, jos järjestelmä muuttuu usein. Muutosanalysoinnin avulla saadaan selville miten ja mitä primitiivi- ja komposiittikomponentteja on muutettava, jos jotain komponenttia muutetaan.

2.5.4 SCM:n suhde sovelluskehityksen hallintavälineisiin

SCM:stä löytyy monia yhteneväisyyksiä versionhallintajärjestelmän ja paketointivälineen toimintoihin ja vaatimuksiin. Useimmat tässä esitellyistä hallinnointitoimista löytyvät jossain muodossa näistä välineistä, jotka esitellään tarkemmin kappaleessa 3.1.

3. SOVELLUSKEHITYKSEN HALLINTAVÄLINEET

3.1 Versionhallinta ja paketointi

3.1.1 Lähdekoodin säilyttäminen

Tärkein syy lähdekoodin keskitettyyn hallintaan on estää tiedon hajautuminen ympäri organisaatiota ja mahdollistaa ohjelmistoyritykselle elintärkeän pääoman varmentaminen eli varmuuskopiointi. Lähdekoodeja myös ylläpidetään useiden henkilöiden toimesta. On tehokkainta, jos heidän tarvitsee käsitellä lähdekoodeja vain yhdestä paikasta käsin.

Lähdekoodin keskitetyllä ja varmistetulla säilyttämisellä pyritään turvaamaan yrityksen toiminta eri tilanteissa eli kyseessä on yritysriskien pienentämiseen tähtäävä toimenpide. Jos yritys on vakuuttanut toimintansa, myös vakuutusyhtiö saattaa vaatia lähdekoodin säilyttämistä ja varmistuksia.

Tästä voidaan päätellä, että kullakin yrityksellä on käytössään jokin versionhallintajärjestelmä, sillä ne mahdollistavat lähdekoodien keskitetyn säilyttämisen, hallinnoimisen ja varmistamisen standardoidulla tavalla.

3.1.2 Versionhallintaohjelmistot

Versionhallintaohjemistoja ovat mm. Concurrent Versions System [CVS], Subversion [Subv] ja Microsoft Visual SourceSafe [VSS] sekä näiden ja monien muiden järjestelmien johdannaiset ja parannellut versioit. Kattava lista erilaisista versionhallintajärjestelmistä löytyy mm. vapaasta tietosanakirjasta Wikipediasta [WkVer].

Versionhallintajärjestelmät on ennen kaikkea suunniteltu merkkimuotoisen datan (esim. lähdekoodi) tallennukseen ja versioimiseen. Monet välineistä tukevat myös ryhmätyöskentelyä mahdollistamalla mm. samaan versioon tehtyjen useamman muutoksen yhdistelemisen (merge) muuttuneiden versioiden viennin yhteydessä.

Tätä tukee mm. CVS. Vaihtoehtoisesti vain yksi käyttäjä pääsee kerrallaan muokkaamaan tiedostoa lukitsemalla sen muokattavaksi vain itselleen. Tämän taas mahdollistaa mm. Microsoft SourceSafe.

3.1.3 Komponenttien versionhallinta

Versionhallintavälineiden tärkein tehtävä on tiedostojen versioiden hallinta. Tämän avulla tiettyyn komponenttiin liittyvät lähdekoodit, resurssitiedostot, konfiguraatiot jne. voidaan merkitä tietyllä versioleimalla (label), jolloin syntyy komponentin versio. Vaikka em. tiedostoihin tehtäisiin muutoksia, on koska tahansa myöhemmin mahdollista koostaa tämä aiempi versio pyytämällä versionhallinnasta tällä versioleimalla merkityt tiedostot. Tästä seuraa, että komponenttien käännöksiä tai niistä koostettuja paketteja (esim. JAR-tiedostoa) ei ole tarve erikseen säilöä mihinkään. Tosin kääntämisessä tarvittavat konfiguraatiotiedot kannattaa säilöä ja versioida samalla itse komponentin lähdekoodin ym. kera.

3.1.4 Komponenttien paketointi

Komponentin koostamista lähdekoodin käännöksestä sekä muista mahdollisista resurssitiedostoista kutsutaan Paketoinniksi. Paketointi käsittää myös asennuspakettien (EAR) koostamisen aiemmin paketoiduista komponenteista. Paketoinnin yhteydessä syntyy aina uusi versio paketoitavasta komponentista.

3.1.4.1 Paketointiväline

Kun em. paketointi tehdään jollain tähän tarkoitukseen luodulla välineellä, säästytään useimmilta inhimillisiltä virheiltä, joita monivaiheisen manuaalisen paketoinnin yhteydessä helposti syntyisi. Paketointiväline hyödyntää monenlaisia konfiguraatiotietoja paketoidessaan uutta komponentin versiota. Konfiguraatiotiedoissa on kuvattu mm. komponentin nimi, lähdekoodin ja muiden resurssitiedostojen sijainti, käytettävä versioleima sekä suorituksen- ja käännöksenaikaiset riippuvuudet muista komponenteista. Paketointiväline leimaa paketoinnin yhteydessä komponentin lähdekoodin ja resurssitiedostot versionhallintajärjestelmässä ko. versioleimalla.

Konfiguraatiotiedot eroavat erityyppisten komponenttien välillä. Esim. WAR- ja EJB-paketteihin liittyy aivan erilainen asennuskuvaimien sijoitteluhierarkia. Lisäksi esim. EJB-komponenttien osalta pitää paketoida myös komponentin palveluja hyödyntävän osapuolen tarvitsema asiakaspaketti (EJB-client).

Paketointivälineen vastuulle on myös parasta jättää kaikki tekninen versiointi ja generoidun komponentin paketin tallennus keskitettyyn paikkaan. Jos generoituja paketteja ei otettaisi talteen, pitäisi kaikki paketoinnissa tarvittavat komponentit aina kääntää uudelleen rekursiivisesti. Tästä saattaisi muodostua tavattoman iso operaatio. Lisäksi kehittäjät voivat hyödyntää käännettyjä paketteja työasemillaan ja integraatiotestiympäristöissään.

3.1.4.2 Paketointivälineen toteutus ja hankinta

Paketointivälinettä ei välttämättä löydy yrityksen tarpeisiin sellaisenaan. Monissa yrityksissä vastaava väline saatetaan toteuttaa itse. Toteutuksessa on erittäin suotavaa hyödyntää kypsiä vapaan lähdekoodin yhteisön luomia apuvälineitä, kuten Ant-työkalua [Ant]. Paketointivälineen pitää ottaa huomioon yrityksessä käytettävät Java EE -palvelinohjelmistot, sillä mm. EJB- ja EJB-client -komponentit käännetään kullekin palvelinohjelmistolle suunnatulla EJB-kääntäjällä (EJB Compiler). Muun muassa edellä mainittu Ant-työkalu auttaa EJB-käännösprosessia [AntEJB] tukemalla useimpia Java EE-palvelimia. Paketointivälineen on tuettava myös yrityksessä käytettyjä versionhallintajärjestelmiä, jotta aiemmin mainittu lähdekoodien, resurssitiedostojen ja konfiguraatiotietojen versiointi on tehtävissä välineen toimesta.

3.1.4.3 Paketointivälineellä tavoitellut hyödyt

Paketointiväline tehostaa komponenttien kehittämistä, koska sen avulla komponenttien konfigurointi, riippuvuuksien kuvaaminen ja kääntäminen ja huomattavasti helpompaa ja nopeampaa kuin käsityönä. Välineen toimesta käännöksestä jää myös tarvittaessa talteen vertailukelpoinen loki esim. virhetilanteiden selvittelyjen avuksi.

Lisäksi välineen avulla voidaan standardisoida käännösympäristö, mikä on edelleen tarpeen, vaikka Java-teknologian pitäisikin olla tästä riippumatonta, mutta käytännössä eri Java-versioiden välillä on eroavaisuuksia. Standardisointi koskee myös joidenkin tuotekomponenttien versioiden yhteistä kiinnittämistä, sillä nämä muodostavat usein tietyn kokonaisuuden eivätkä toimi oikein, jos eri kokonaisuuksien versioita sekoitetaan keskenään. Tämä on eritoten toteutunut Javalla toteutettujen XML-käsittelyyn keskittyneiden komponenttikokonaisuuksien kohdalla. Näiden komponenttien tietyt versiot ovat tiukasti nivoutuneita toisiinsa ja heikosti hallinnoituna aihettavat herkästi erikoisia ongelmia.

3.1.4.4 Paketointivälineen ja komponenttikuvauskannan yhteistyö

Komponenttikuvauskannan tietojen tulisi olla aina ajan tasalla. Välineen tulisi olla tietoinen kaikista yrityksessä olevista komponenteista ja niiden versioista. Tämän vuoksi paketointivälineen ja komponenttikuvauskannan kannattaa tehdä yhteistyötä, vaikka ensin mainittu onkin operatiivinen ja jälkimmäinen informatiivinen sovelluskehityksen hallintaväline. Yhteistyö voidaan toteuttaa esim. mahdollistamalla komponenttien kuvaamisen myös paketointivälineellä, joka sitten välittää tiedot komponenttikuvauskantaan. Eritoten aina kunkin paketoinnin yhteydessä syntyvän version tiedot olisi hyödyllistä siirtää ajantasaisesti komponenttikuvauskantaan.

Integraatio on kuitenkin syytä toteuttaa löyhästi, sillä paketointivälineen toiminta ei saisi häiriintyä millään tavalla, vaikka komponenttikuvauskannan palvelut eivät olisikaan aina saatavilla. Ajantasaisesti siirtämättä jääneet tiedot on mahdollista viedä myöhemminkin komponenttikuvauskantaan.

3.2 Komponenttikuvauskanta

> *"Field of Dreams. This pitfall derives from the mistaken belief that if you "build a repository of components, reusers will come.""*
>
> *Barry Boehm, Managing software productivity and reuse [Boe99]*

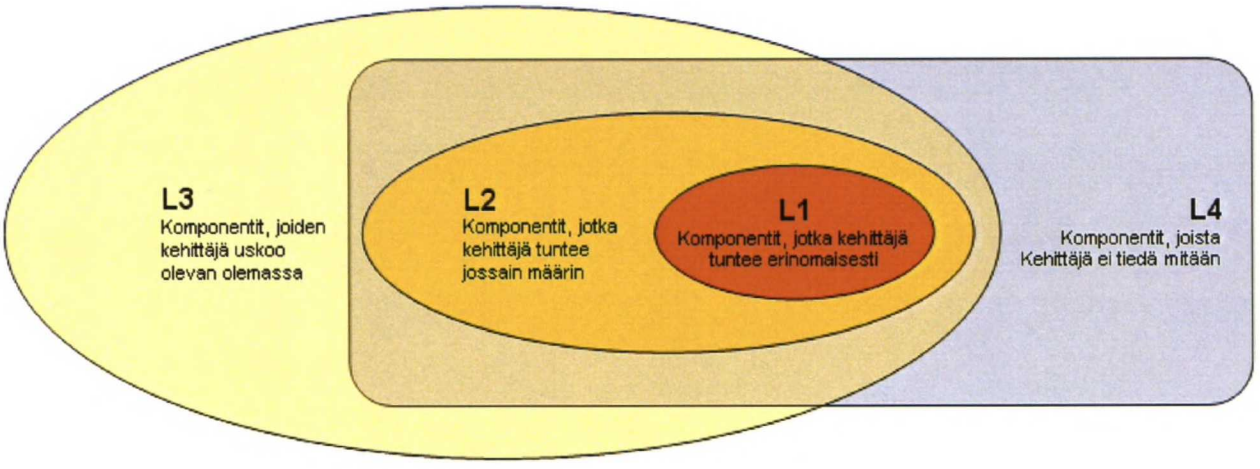
Lähdekoodin keskittäminen ei riitä siihen, että sovelluskehittäjät löytäisivät tietyn toiminnallisuuden tarjoavan komponentin riittävän helposti tai nopeasti. Tätä varten yrityksessä on hyvä olla eräänlainen kirjasto, jonka kautta voi tutkia kaikkia olemassa olevia komponentteja. Kirjastoanalogia on seurausta siitä, että komponenteista pidetään samaan tapaan yllä metatietoja, kuten kirjaston tuotteista on olemassa kortistot ja tietokannat. Tätä kirjastoa voidaan kutsua komponenttikuvauskannaksi. Hyvin toteutetuilla komponenttien paikannuspalveluilla saavutetaan lisää tehokkuutta sovelluskehittämiseen. Komponenteista kuvattavia metatietoja käsitellään tarkemmin kappaleessa 6.1. Komponenttikuvauskannan yksi perusidea on komponentín käyttämiseksi tarvittavan informaation erottaminen toteutuksen yksityiskohdista.

3.2.1 Tiedotus ja viestintä

Komponenttikuvauskanta mahdollistaa uuden tavan komponenteista tiedottamiseen, jota voidaan myös osittain automatisoida. Ilman tiedotusta tieto komponenteista leviää hallitsemattomasti kehittäjältä toiselle. Walton sekä Rosenbaum et al. [Wal92, Ros95] toteavat yhden olennaisimman uudelleenkäyttöä edistävän tekijän olevan juuri uudelleenkäytettävien komponenttien tunnetuksi tekeminen. Heidän mukaansa komponenttien mainostaminen ei kuitenkaan yksinään riitä, vaan komponenteista pitää olla myös oikeanlainen dokumentaatio saatavilla.

Komponenttikuvauskannassa käytetty terminologiaa voidaan hyödyntää myös kaikessa yrityksen sisäisessä viestinnässä. Tämä tietenkin edellyttää, että terminologia on käytännöllinen ja hyvin suunniteltu. Yhteistä terminologiaa käyttämällä vältytään helpommin väärinkäsityksiltä, mikä todennäköisesti tehostaa sovelluskehitystä sujuvamman viestinnän seurauksena.

Tiedottamisella voidaan myös vähentää Yunwen Ye:n ja kumppaneiden [Ye00] kuvaamaa tietosaarekkeiden (information islands) muodostumista sovelluskehittäjien keskuudessa. Ye et al. [Ye00] jakavat sovelluskehittäjien komponenttien käytön kolmeen kategoriaan: muistaminen (reuse-by-memory), muisteleminen (reuse-by-recall) ja ennakoiminen (reuse-by-anticipation). Muistaminen vastaa tilannetta, jossa kehittäjä uutta sovellusta tai komponenttia toteuttaessaan tunnistaa tarpeen toiminnallisuudelle, jonka hän muistaa olevan toteutettu jossain toisessa komponentissa ja jonka hän voi helposti ottaa käyttöön muistinvaraisesti. Muisteleminen eroaa edellisestä siinä mielessä, että kehittäjä muistelee komponentin olevan olemassa, mutta tarvitsee komponenttikuvauskannan palveluita löytääkseen juuri oikean komponentin. Ennakoimisen yhteydessä kehittäjä olettaa jonkun toiminnallisuuden olevan toteutettuna ja yrittää varmistua tästä etsimällä vastaavaa toteutusta komponenttikuvauskannasta. Näiden ulkopuolelle jäävät komponentit, joiden olemassaoloa kehittäjät eivät tiedä, muista tai edes osaa ennakoida. Seuraavassa kuvassa [Kuva 4] on havainnollistettu sovelluskehittäjien komponenttien tietämystasot. Kuvassa taso L1 vastaa kategoriaa muistaminen, L2 kategoriaa muisteleminen ja L3 kategoriaa ennakoiminen. Komponentit tasossa L4-L3 muodostavat aiemmin mainitut tietosaarekkeet.



Kuva 4 – Sovelluskehittäjien komponenttien tietämystasot

3.2.2 Komponenttien fyysinen varastointi

Komponenttikuvauskannan perustaminen ei sellaisenaan vaadi uuden keskitetyn komponenttivaraston perustamista. Mutta sellaisen perustamista on syytä harkita, jos käännetyt komponentit ovat siihen asti sijainneet monessa eri paikassa. Keskittäminen helpottaa mm. varmennusta ja käyttöönottoa kehittäjien suoritusympäristöissä sekä nopeuttaa paketointia [3.1.4.1]. Tämä ei tarkoita, että komponentit tallennettaisiin samaan yhteyteen kuin itse komponenttikuvauskanta. Tämä ei ole suotavaa, koska informatiiviset ja operatiiviset järjestelmät on syytä pitää erillään. Komponenttien muodostamiseen tai noutamiseen voidaan käyttää esim. paketointivälinettä. Tällöin komponentit voidaan haluttaessa varastoida useampaankin paikkaan, koska niitä on mahdollista noutaa käyttöönsä yhden välineen kautta.

3.2.3 Kehittäjälähtöisyys

Kukin käyttää Komponenttikuvauskannan palveluita omalla tavallaan. Tämän vuoksi komponenttien paikantamiseen on hyvä tarjota vaihtoehtoja. Mahdollisia paikannustapoja ovat esim. erilaiset selailunäkymät sekä hakutoiminnot. Näiden toimintojen tulee olla helppoja ja nopeita, jotta oikeasti saavutettaisiin tehokkuutta kehittämiseen [Ye00]. Myös uusien komponenttien lisääminen tulee käydä helposti ja ilman että käyttäjän tarvitsee syöttää tietoja moneen kertaan. Komponenttikuvauskantahan merkitsee joka tapauksessa jonkin verran lisätyötä kehittäjille komponenttien kuvaamisen muodossa. Samalla kuitenkin kuvaamiseen panostettu työ säästetään moninkertaisesti, koska se on tehtävä vain kertaalleen eikä sitä tarvitse viestiä erikseen kullekin komponentista kiinnostuneelle taholle.

Ye et al. [Ye00] väittävät, että komponenttikuvauskanta tulee suunnitella kehittäjälähtöisestä näkökulmasta, jos halutaan kehittäjien omaksuvan komponenttikuvauskannan pysyvästi osaksi omaa sovelluskehitysprosessiaan. Yrityksen tasolla yksi komponenttikuvauskannan käyttämisen päämotivaatioista on uudelleenkäytön lisääminen ja sitä kautta sovellusten laadun ja tuottavuuden parantuminen. Kehittäjät eivät ajattele näin korkealentoisesti kehitystyötä tehdessään. Heidän onkin itse nähtävä ja koettava uudelleenkäytön välittömät hyödyt.

Uudelleenkäytön omaksumisessa ja lisäämisessä yksittäisen kehittäjän tasolla auttaa, jos komponenttikuvauskanta on toteutettu tukemaan ennen kaikkea heidän arkitarpeitaan. [Ye00]

3.2.4 Komponenttien luokittelu

Jotta selailu- ja hakutoiminnot voitaisiin toteuttaa järkevästi, tulee komponentteja voida luokitella. Luokittelu voi olla myös kevyt, jotta se ei aiheuttaisi tarpeetonta päänvaivaa komponentteja lisättäessä. Luokittelua määriteltäessä on hyvä tiedostaa, ettei suunniteltu luokittelu välttämättä sovellukaan sellaisenaan jatkossa. Siksi luokittelu kannattaa toteuttaa siten, että sen muuttaminen käy kohtuullisen helposti. Käyttökokemusten ja havaittujen ongelmien myötä luokittelu jalostuu käyttäjien mielessä. Tällöin on järkevää päivittää myös komponenttikuvauskannan luokittelu sen mukaiseksi. Tämän vuoksi kannattaa suosia toteutuksessa jonkinlaista inkrementaalista luokittelua [Hen96], jolloin luokittelua voidaan myöhemmin syventää tai laventaa tarvittaessa uusilla vaihtoehdoilla ilman koko järjestelmää koskevia muutoksia. Henningerin mukaan turhan usein luokittelu kiinnitetään alussa, jolloin sen muuttaminen vaatii koko kuvauskannan uudelleensuunnittelua. Tämä on hänestä merkittävä ongelma – varsinkin, kun luokittelua on äärimmäisen vaikea saada kuntoon kerralla. Luokittelu onkin täten syytä aloitaa hitaasti ja sen kehittäminen parasta hoitaa iteratiivisesti. Henninger [Hen96] ehdottaa myös hyödynnettävän adaptiivisia tekniikoita, jolloin komponenttikuvauskanta kirjaa käyttäjien toimia ja muuntaa toimintaansa havaintojensa perusteella.

Luokittelua voidaan käyttää hyväksi myös aiemmin mainitussa automatisoidussa tiedottamisessa. Esim. kaikista yleiskomponenteista voidaan lähettää automaattisesti tiedotus sovelluskehittäjille tai niistä voidaan luoda oma lokinsa, jonka kautta kehittäjät voivat seurata viimeisimpiä yleiskomponentteihin tehtyjä muutoksia.

Luokittelu on myös ensimmäinen tekijä, kun kehittäjä arvioi komponentin soveltuvuutta kehitteillä olevan sovelluksensa käyttöön. Eli hyvin suunniteltu luokittelu tehostaa kehittäjien komponenttien arviointiprosessia.

3.2.5 Komponenttien kartoitus

Komponenttikuvauskannan käyttöönottoa edeltää aina olemassa olevien komponenttien kartoitus ja luokittelu. Kuvauskannasta on huomattavasti enemmän hyötyä jos käytännössä kaikki yrityksestä löytyvät komponentit on kuvattu siellä. Jo kartoitusvaiheessa on syytä ajatella kerätyn tiedon automatisoitua vientiä kuvauskantaan. Esim. Word-dokumentti ei ole yhtä hyvä idea kuin vaikkapa Excel-taulukko, jonka sisältö on purettavissa monilla eri tavoilla. Kuitenkaan liian raskasta tekniikka ei myöskään kannata soveltaa. Esim. tietojen syöttäminen kartoitusvaiheessa suoraan tarkoitusta varten luotuun relaatiotietokantaan vaikuttaa ylilyönniltä.

3.2.6 Interaktiivinen kehitysympäristö

Ye et al. [Ye00] ehdottavat puoliautomaattista aktiivista tiedonvälitystekniikkaa, jossa komponenttikuvauskanta on integroitu sovelluskehitysympäristöön. Kehittäjän työskentelyn aikana taustajärjestelmät muodostavat kyselyitä komponenttikuvauskantaan ja tarjoavat käyttäjälle tietoa uudelleenkäytettävistä toteutuksista. Nämä taustajärjestelmät tarkkailevat kehittäjän kirjoittamaa lähdekoodia ja sen kommentointia, joiden pohjalta ne muodostavat kyselynsä. Ye:n mukaan tämä "rakentaa sillan tietosaarekkeille" tarjoamalla kehittäjälle tietoa hänelle aiemmin tuntemattomista komponenteista.

Ehdotuksen soveltaminen tämän tutkimuksen komponentti-käsitteen yhteydessä ei ole kuitenkaan helppoa. Ye et al. [Ye00] käsittävät komponentilla huomattavasti matalamman tason elementtejä, kuten yksittäisiä luokan metodeja, joiden puumerkkeihin (signature) aktiivinen kyselyiden muodostaminenkin perustuu. Kuitenkin itse idea aktiivisesta kytkennästä kuvaustietojen ja kehitysvälineiden välillä kiehtoo suuresti. Sovelluskehitysvälinevalmistajien (esim. Borland ja IBM) soisi enenevässä määrin sisällyttävän tämän kaltaista toiminnallisuutta tulevaisuuden sovelluskehitysvälineiden suunnitelmiinsa. Sovelluskehitykseen käytettävissä työasemissa on nykypäivänä niin paljon käyttämätöntä kapasiteettia, etteivät moiset taustaprosessit juuri hidastaisi kehitysympäristöä.

3.3 Konfiguraation hallinta

Konfiguraation hallinnan toiminnot on suunnattu lähinnä suoritusympäristöjen hallintaan. Konfiguraatiolla tarkoitetaan tässä yhteydessä tietystä suoritusympäristöstä löytyviä komponentteja ja sovelluksia. Konfiguraation hallinnan yhteydessä sovellusten asennuspaketit eli EAR-paketit ovat erittäin tärkeässä osassa.

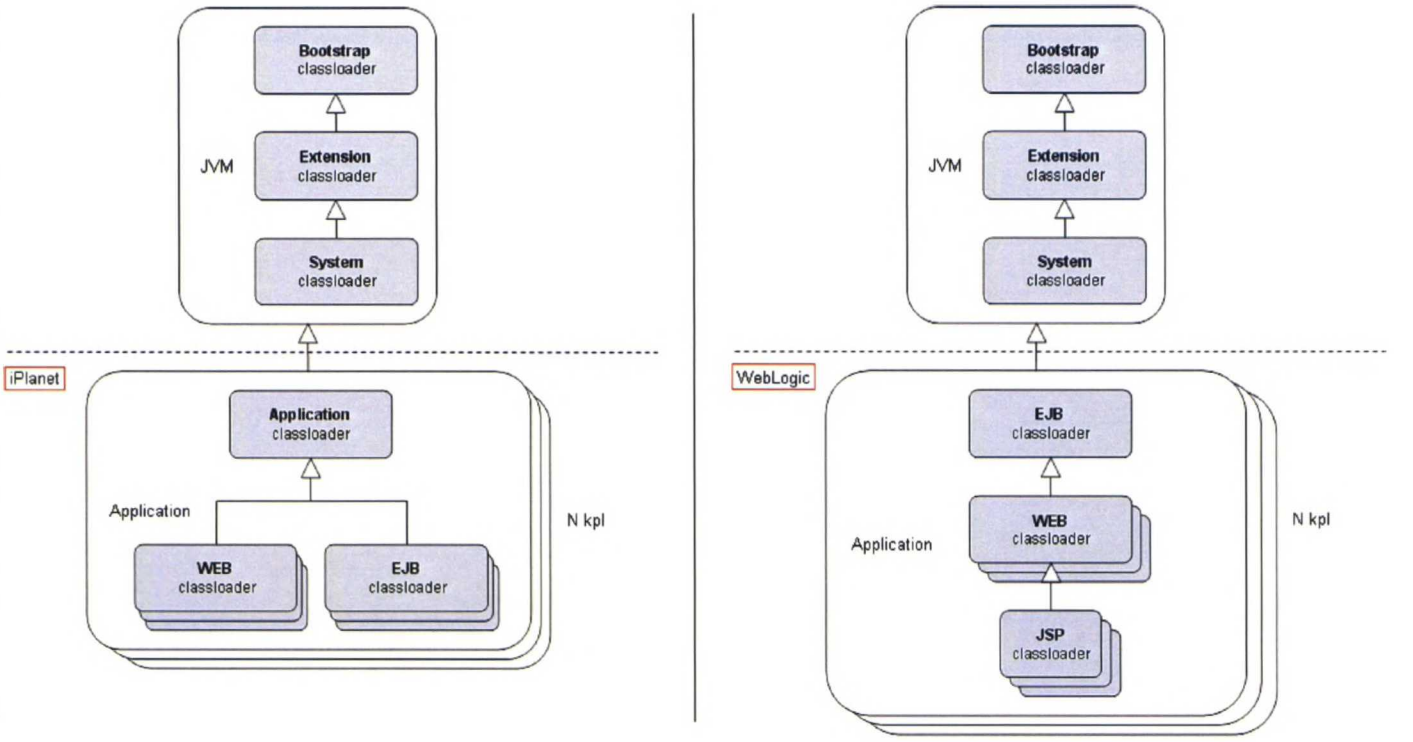
Konfiguraation hallinta -välineellä on mahdollista tarkastella kaikkia suoritusympäristöjä ja tehdä tarvittaessa muutoksia eli lisätä, päivittää tai poistaa komponentteja ja asennuspaketteja.

3.3.1 Luokkien lataaminen sovelluspalvelimissa

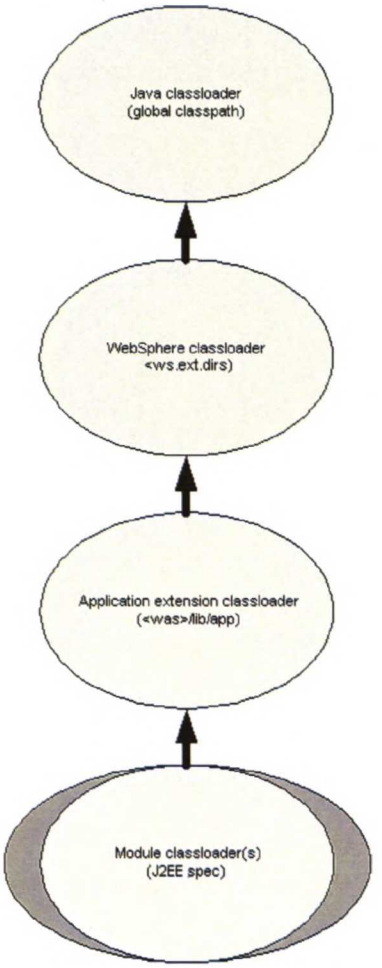
Konfiguraationhallintavälineestä on löydyttävä tuki yrityksessä käytössä oleville sovelluspalveliminohjelmistoille, joilla kullakin on oma komponenttien asennus- ja sijoittelumallinsa sekä oma tapansa ladata muistiinsa komponenttien sisältämät Java-luokat suorituksen aikana.

Komponentin tyypistä riippuen niitä voidaan asentaa yhteen suoritusympäristöön eri tasoille, joilla on toisiinsa perintöhierarkiasuhde (parent-child). Tarkempi sijoittelu on pitkälti palvelinohjelmistosta kiinni. Eri palvelinohjelmistot lataavat komponenttien sisältämät Java-luokat eri ClassLoader:eihin oman sisäisen luokkahierarkiamäärittelynsä mukaisesti. Pääsääntöisesti ne perivät Java Virtuaalikoneen sisäisen luokkapolun ja luovat sen päälle oman luokkienlataushierarkiansa. Muutama yleistävä esimerkki kolmen sovelluspalvelimen (iPlanetin, WebLogicin ja WebSpheren [iPlaCL, WLSCL, WebSp]) ClassLoader-hierarkiasta on nähtävissä seuraavissa kuvissa [Kuva 5, Kuva 6].

Sovelluspalvelimet sisältävät joskus omia erikoisuuksiaan, kuten BEA WebLogic, jossa Web-sovelluksen (WAR-komponentti) osalta on mahdollista käskeä palvelinohjelmistoa suosimaan sen sisältämiä luokkia ja komponentteja (prefer-web-inf-classes). Näin tietyn Web-sovelluksen sisällä on mahdollista käyttää eri versiota luokista, jotka ovat esim. WebLogicin tai Java virtuaalikoneen sisäisesti käyttämiä.



Kuva 5 – iPlanet- ja WebLogic-sovelluspalvelimien ClassLoader-hierarkiat



Kuva 6 – WebSphere ClassLoader-hierarkia

3.3.2 Komponenttien päivittäminen suoritusympäristössä

Yksi suoritusympäristön hallinnan vaikeimmista tehtävistä on tietyn palvelun (esim. EJB) päivittäminen s.e. kaikki sitä kutsuvat sovellukset saavat sen heti käyttöönsä ilman ajonaikaisia virheitä, joita mm. eri versiot palvelun rajapinnassa välitettävien luokkien osalta saattavat aiheuttaa. On siis varmistettava, että kaikilla palvelua kutsuvilla sovelluksilla on käytössään oikean, kulloinkin asennetun, palveluversion asiakasrajapintatoteutusluokat (EJB-palvelun tapauksessa EJB-client). Tähän on ainakin kaksi vaihtoehtoa. Ensinnäkin EJB-client -komponentti on mahdollista suoritusympäristön luokkapolkuun, josta se tulee automaattisesti käyttöön kaikissa sitä kutsuvissa sovelluksissa. Toinen vaihtoehto on rakentaa konfiguraationhallintavälineeseen tuki prosessille, jossa paketoidaan ja asennetaan kaikki päivitettävää palvelua kutsuvat sovellukset uusiksi s.e. niihin sijoitetaan oikean version asiakasrajapintatoteutusluokat. Jälkimmäisellä säästytään

luokkapolkuun sijoitettavilta kirjastoilta ja palvelimen turhalta alasajolta päivitysten yhteydessä, mutta samalla otetaan uusi riski tekemällä päivityksiä "täysin toimiviin" sovelluksiin vain siksi, että yksi sen käyttämä palvelu on päivitettävä. Konfiguraation hallinta -väline ei siis ratkaise tätä ongelmaa yrityksen puolesta, mutta sen tulee kylläkin tukea teknisesti yrityksessä käytettävää päivitysprosessia. Prosessi tehostaisi komponenttien useiden eri versioiden hallintaa automatisoiden päivitysprosessin. Väline tekee sen luotettavammin ja vähemmillä virheillä kuin jos päivitysprosessi suoritettaisiin manuaalisesti.

4. KOMPONENTTIKUVAUSKANNAN TOTEUTUSMALLI

4.1 Toteutusmallin muodostuminen

Tutkimuksen päätavoitteena on löytää kohdeyrityksen sovelluskehitysprosessiin istuva toteutusmalli komponenttikuvauskannalle. Tutkimuksen aikana on käynyt jatkuvasti selvemmäksi, että valmista mallia ei olisi löydettävissä akateemisesta tai teollisesta maailmasta. Myöskään kohdeyritykseen soveltuvaa komponenttikuvauskantavälinettä tai sen sisältävää hallintavälinekokonaisuutta ei ole tullut vastaan tutkimuksen aikana. Koska referenssimallia ei ole löytynyt, toteutusmalli on muodostettava kohdeyrityksen omien sisäisten tarpeiden mukaisesti. Toteutusmalliin vaikuttavat eritoten kohdeyrityksen sovelluskehitysprosessi, käytettävät teknologiat sekä yrityksestä jo löytyvät tai valmisteilla olevat muut sovelluskehityksen hallintavälineet.

4.2 Toteutusmallivaihtoehdot

4.2.1 Osa laajempaa yhtenäistä hallintavälinetuotekokonaisuutta

Yrityksen on mahdollista ottaa käyttöön isompi yhtenäinen tuotekokonaisuus ja hyödyntää sen tarjoamia komponenttikuvauskantapalveluja. Tämä toteutusmalli vaatii mahdollisesti suuria muutoksia sovelluskehitysprosessiin. Tutkimuksen kohdeyrityksessä on jo pitkälle organisaatioon jalkautuneita sovelluskehityksen hallintavälineitä, joiden korvaaminen isomman tuotekokonaisuuden tarjoamilla välineillä ei olisi halpaa, helppoa tai edes hyödyllistä. Kohdeyrityksen hallintavälineet on optimoitu kehitysprosessin tarpeiden mukaisiksi. Lisäksi käyttäjät ovat jo omaksuneet kunkin välineen monipuolisen toiminnallisuuden.

4.2.2 Itsenäinen komponenttikuvauskantatuote

Onkin helpompaa toteuttaa tai ottaa käyttöön yksinomaan komponenttien kuvaamiseen tarkoitettu tuote ja integroida se muihin olemassa tai kehitteillä oleviin hallintavälineisiin. Tietenkin komponenttikuvauskanta voi hyödyntää toteuksessaan

muita tuotteita, varsinkin teknisesti haastavampien tai pitkälle standardoitujen tarpeiden tyydyttämiseksi. Esimerkki tällaisesta tarpeesta on muutosten vaikutusalueen kartoitus (Impact Analysis), jota käsitellään tarkemmin kappaleessa 6.2.8.

4.2.3 Metatietokuvauskannat

Jos yrityksessä koetaan hankalaksi toteuttaa komponenttikuvauskantatuote itse, voidaan komponenttien kuvaamisessa hyödyntää erilaisten metatietojen kuvaamisen mahdollistavaa tuotetta. Näillä sisällönhallintatuotteilla on mahdollista kuvata hyvin erilaisia tietojoukkoja. Tämän toteutusmallin yksi etu on saman kuvaustuotteen potentiaali myös muiden kuvauskantojen teknisenä alustana. Esim. palvelu- ja prosessikuvauskanta ovat samantyyppisiä komponenttikuvauskannan kanssa. Ne kuvaavat vähemmän teknisiä asioita, mutta sisällönhallintatuote kyllä soveltuu yhtä lailla näidenkin kuvaamiseen. Kaikki oikeastaan riippuu sisällönhallintatuotteen konfiguroitavuudesta eli siitä, miten sillä kuvattavien tietojen sanasto on määriteltävissä.

Esimerkki tällaisesta metatietokuvauskantatuotteesta on Profium-yrityksen SIR-tuote [SIR]. SIR on yksi ensimmäisistä sisällönhallintajärjestelmistä, jotka nojaavat yksinomaan yleisesti hyväksyttyihin avoimiin standardeihin. Profiumin mukaan standardit takaavat patentoituja ratkaisuja järkevämmän kehityssuunnan tulevaisuuden tarpeisiin mukautumisessa. SIR-tuotteessa hyödynnetään metatietojen kuvaamiseen W3C-organisaation Resource Description Framework -standardia [RDF], joka mahdollistaa vapaavalintaisiin resursseihin liittyvän metatiedon esittämisen WWW-teknologian avulla. RDF:n metatietosanasto esitellään RDF Schema -kuvauskielen avulla. Komponenttikuvauskannan tapauksessa sanastolla määriteltäisiin komponentista kuvattavat tiedot ja niiden rajoitteet.

4.3 Sovelluskehityksen hallintavälineet kohdeyrityksessä

4.3.1 Versionhallintajärjestelmät

Kohdeyrityksessä on käytössä muutama eri versionhallintajärjestelmä, joista kunkin sisältö painottuu tietyillä ohjelmointikielillä tai -teknologioilla toteutettuihin ohjelmiston osiin. Esim. lähes kaikki Java-lähdekoodi ja siihen liittyvät resurssitiedostot sekä paketointivälineellä koostettujen komponenttien paketit tallennetaan Microsoft SourceSafe -versionhallintajärjestelmään.

4.3.2 Paketointiväline

Kohdeyrityksestä löytyy myös omaan käyttöön toteutettu Java-komponenttien paketointiväline. Nykyisin, ennen komponenttikuvauskannan olemassa oloa, paketointivälineellä kuvaillaan lyhyesti kukin komponentti sekä kuvataan käännöksen ja suorituksen aikaiset riippuvuudet muista komponenteista. Paketointivälineen vastuulla on myös Javadoc-rajapintadokumentaation generoiminen, koska oikein suoritettuna se vaatii samat riippuvuudet kuin lähdekoodin käännöksen aikanakin.

Paketointiväline hyödyntää tietovarastoinaan edellä mainittua Microsoft SourceSafe-versionhallintajärjestelmää ja verkkolevyresursseja. Paketointiväline tallentaa kehittäjien työasemilla paketoidut komponenttien paketit sekä kunkin komponentin paketointiin liittyvät konfiguraatiotiedot versionhallintajärjestelmään. Paketoijien paketoimat komponetit väline tallentaa verkkolevyresurssille. Paketointivälineellä muodostetaan myös komposiittikomponentteja eli EAR-paketteja, jotka koostetaan JAR-, EJB-, WAR- ja RAR-paketeista.

4.3.3 Konfiguraation hallinta -väline

Komponenttikuvauskanta-projektin rinnalla käynnissä olleen toisen projektin vastuulla oli prototyypin pohjalta tuotteistaa ja ottaa käyttöön konfiguraation hallinta -väline. Välineen avulla on mahdollista välttää asennuksissa tapahtuvia inhimillisiä virheitä sekä automatisoida yksinkertaisimpia asennuksia. Lisäksi sen kautta on nähtävissä kaikki kohdeyrityksen suoritusympäristöt konfiguraatiotietoineen.

Konfiguraatiolla sisältää mm. suoritusympäristöön asennetut komponentit ja sovellukset. Konfiguraation hallinta -välinettä käyttävät eritoten paketoija sekä asentajat asennusten tilaamiseen ja suorittamiseen. Heidän lisäkseen kuka tahansa voi tutkia sen kautta yrityksen Java EE -suoritusympäristöjä. Yleiskäyttöisten komponenttien vastuuhenkilöiden on myös mahdollista nähdä missä kaikkialla heidän vastuullaan olevia komponentteja käytetään. Konfiguraation hallinta -välinettä ollaan ottamassa käyttöön suunnilleen samassa aikataulussa kuin komponenttikuvauskantaa.

4.4 Komponenttien uudelleenkäyttö kohdeyrityksessä

Kohdeyritys kuuluu sovelluskehitystä tekevien yritysten suureen enemmistöön siinä mielessä, että yrityksessä ei toteudu CBSD-mallin tasoinen komponenttien uudelleenkäyttö. Tämä johtuu mm. yritysten ja koko yhteiskunnan nykytrendin mukaisesta projektivetoisuudesta. Yksikään projekti ei halua tuottaa uudelleenkäytettäviä komponentteja, vaan projektien tärkein tavoite on tyydyttää liiketoiminnan tarpeet mahdollisimman halvalla ja nopeasti. Aidosti komponenttipohjaisen sovelluskehitysmallin käyttöön ottaminen vaatisi huomattavia panoksia sekä rahallisesti että ajallisesti. CBSD-mallin ongelmia ja esteitä käsiteltiin tarkemmin kappaleessa 2.4.4.

Kohdeyrityksessäkin kehitys painottuu liiketoiminnan tarpeista käynnistyviin projekteihin, jotka hyödyntävät olemassa olevia yleispalveluita. Aika ajoin näiden projektien rinnalle perustetaan uusia yleispalveluja kehittäviä rajapinta- ja teknologiapainotteisia projekteja. Tämä johtaa osittaiseen uudelleenkäyttöön, mutta edelleen monissa projekteissa toteutetaan asioita itse, vaikka sama toiminnallisuus olisikin toteutettu jo aiempien projektien yhteydessä. Joskus projektien sisällä omiin tarpeisiin tuotetut yleisiksi miellettävät toteutukset tuotteistetaan, milloin niistä luodaan oma yleiskomponenttinsa tai ne liitetään osaksi jotain olemassa olevaa yleiskomponenttia. Myöskään kohdeyrityksen kaikki bisnesongelmat eivät ole helposti komponentoitavissa, mikä olisi yksi edellytys komponenttien laajamittaiselle uudelleenkäytölle.

Kohdeyrityksen komponentit eivät siis ole pääosin uudelleenkäytettäviä, lukuun ottamatta erityisiä yleispalveluja ja kolmannen osapuolen komponentteja.

Komponenttikuvauskanta voikin vastata lähinnä tiedotus-, viestintä-, ohjeistus- ja uudelleenkäytön edistämistarpeesiin. Komponentteihin liittyvää tiedotusta ja viestintää on nykyisellään hyvin vähän tai ei lainkaan, joten se on kohdeyrityksen osalta kaikkein tärkein yksittäinen komponenttikuvauskannan tehtävä. Ohjeistustarve on myös ilmeinen, sillä varsinkin uudelleenkäytettävistä komponenteista on tähän asti pitänyt pyytää ohjeita ko. komponenttien vastuuhenkilöiltä. Tämä toistuu käytännössä lähes jokaisen yleispalvelua hyödyntävän projektin kohdalla. Uudelleenkäyttöä komponenttikuvauskanta edistää lähinnä tarjoamalla kehittäjille mahdollisuuden tutkia, josko hänen suunnittelemaansa toiminnallisuutta olisi toteutettu jo jossain olemassa olevassa komponentissa tai käynnissä olevassa projektissa.

4.5 Komponenttien hallinta kohdeyrityksessä

Komponenttien useiden versioiden hallinnan helpottamiseksi kohdeyrityksessä on toteutettu jo hyvissä ajoin paketointiväline. Versioiden välinen koordinointi on ollut enemmänkin kiinni kustakin komponentista ja sen vastuuhenkilöistä. Tämän koordinoinnin helpottamiseksi paketointivälineellä kuvattavia käännöksen ja suorituksen aikaisia riippuvuuksia on tulevaisuudessa mahdollista etsiä myös käänteisessä järjestyksessä. Tämän haun perusteella kunkin vastuuhenkilön on mahdollista tarkistaa mistä kaikista komponenteista on riippuvuus hänen vastuullaan olevaan komponenttiin. Tällöin hänen on helpompi koordinoida näiden päivittämistä ja regressiotestausta. Tämä ei kuitenkaan korvaa muutosten vaikutusalueen kartoitus -toiminnallisuutta, jonka avulla regressiotestauksesta saadaan mielekkäämpää ja tarkempaa. Muutosten vaikutusalueen kartoitusta käsitellään tarkemmin kappaleessa 6.2.8.

Myös kohdeyrityksen konfiguraation hallinta -välineestä on apua komponenttien hallinnoinnissa. Sen kautta vastuuhenkilö näkee missä hänen vastuullaan olevaa komponenttiansa käytännössä hyödynnetään. Välineen avulla on myös mahdollista päivittää automaattisesti esim. EJB-komponentin asiakaspään paketti (EJB-client) kaikkien niiden sovelluksien EAR-paketteihin, jotka ko. palvelua hyödyntävät [3.3.2].

4.6 Oma toteutusmalli

Kohdeyrityksen paketointiväline on muodostunut erittäin olennaiseksi osaksi sovelluskehitysprosessia eikä siitä olisi ollut järkevää luopua. Koska paketointiväline on toteutettu kohdeyrityksen omasta toimesta, on sen integroiminenkin helpompaa oman komponenttikuvauskantatoteuksen yhteydessä. Omaan toteutusmalliin päädyttiin myös siksi, että oman toteutuksen kautta on mahdollista luoda komponenttien kuvailutietojoukosta juuri sellainen kuin kohdeyrityksessä on haluttu.

Lisäksi oman komponenttikuvauskantasovelluksen toteutus on kohtuullisen pieni työ, sillä sovellusproblematiikaltaan järjestelmä on melko yksinkertainen. Suurimmat haasteet liittyivät hakutoiminnallisuuteen. Haku mielletään komponenttikuvauskannan tärkeimmäksi toiminnoksi, jolle asetetuista vaatimuksista ei missään nimessä haluta tinkiä.

Oma toteutusmalli jakautuu kahteen käyttökanavaan: WWW-selaimeen ja paketointivälineeseen. Myös paketointiväline integroidaan komponenttikuvauskantaan WWW-teknologioiden avulla (HTTP, Java Servlet, XML).

Selaimen kautta on mahdollista suorittaa kaikkia muita komponentteihin liittyviä hallintatoimenpiteitä paitsi luoda komponenteista uusia versioita, mikä on jätetty paketointivälineen vastuulle. Paketointiväline toimittaakin uusien versioiden tiedot komponenttikuvauskantaan paketoinnin yhteydessä. Näihin versiotietoihin sisältyy myös paketointivälineen generoima Javadoc-rajapintadokumentaatio.

Paketointivälineen pääasiallinen operatiivinen tehtävä on mahdollistaa uusien komponenttien luonti ja uusien versioiden paketointi. Kummankin tehtävän yhteydessä on mahdollista päivittää komponentin kuvausta komponenttikuvauskannassa. Selaimen kautta on mahdollista lisätä komponentti kuvauskantaan jo etukäteen, jolloin paketointivälineellä uutta komponenttia luotaessa haetaan pohjaksi aiemmin syötetyt kuvailutiedot komponenttikuvauskannasta. Etukäteen tehty lisäys voidaan mieltää yhdeksi tiedotustavaksi. Sen kautta kehittäjä voi ilmoittaa muille tietyn komponentin olevan kehitteillä. Lisäksi tällä tavalla on mahdollista kuvata komponentteja, joita ei koskaan paketoida paketointivälineellä.

5. KOHDEYRITYKSEN SOVELLUSKEHITYSPROSESSI

5.1 Tarkoitus

Koska toteutusmalli perustuu kohdeyrityksen sisäisiin tarpeisiin, on kuvattava kohdeyrityksessä ylläpidettävä vakiintunut ohjeistetettu sovelluskehitysprosessi. Prosessiin kuuluvat työtehtävät on roolitettu. Roolien avulla määritellään kaikkien osapuolien vastuut ja tehtävät. Sovelluskehityksen hallintavälineisiin liittyy myös monenlaisia vastuita ja tehtäviä, jotka ovat liitettävissä näihin sovelluskehitysprosessin rooleihin. Välineisiin liittyy myös omia erityisroolejaan, joiden vastuulla on erityisesti ylläpito- ja tukitehtäviä.

Sovelluskehitysprosessiin liittyy myös olennaisesti kohdeyrityksen toimintaympäristö, jonka puitteissa roolien vastuulle kuuluvia tehtäviä suoritetaan. Toimintaympäristön kuvaaminen selkiyttää roolien välisiä suhteita ja tiedon kulkua organisaatiossa.

5.2 Roolit

Seuraavassa taulukossa [Taulukko 1] esitellään kohdeyrityksen sovelluskehitysprosessiin osallistuvien roolien vastuulle kuuluvat tehtävät.

Taulukko 1 – Sovelluskehityksen roolit

Rooli	Tehtävät
Kehittäjä	✓ Kehittäjän tehtäviin kuuluu sovelluksen toteutuksen suunnittelu sekä toteutus hänelle toimitetun toiminnallisen määrittelyn mukaisesti. ✓ Kehittäjän tulee myös testata toteuttamansa sovelluksen komponentit ja luokat yksikkötestauksessa. ✓ Kehittäjä versioi testatut lähdekoodi- ja resurssitiedostot sekä leimaa komponentit (JAR, EJB, WAR ja RAR) ja sovelluspaketit (EAR) paketoinnin yhteydessä. ✓ Kehittäjä kuvaa toteuttamansa komponentit

	komponenttikuvauskantaan ja pitää tietoja jatkossa yllä muutosten yhteydessä. ✓ Kehittäjä työskentelee tiiviissä yhteistyössä paketoijan kanssa, jotta sovellus saadaan muodostettua tarpeellisella tietämyksellä. ✓ Kehittäjä paketoi valmiin sovelluksen integraatiotestivaiheessa ja testaa sovelluksen toiminnan yhdessä paketoijan kanssa. ✓ Kehittäjä antaa paketoijoille herätteen aina, kun sovellus pitää siirtää järjestelmätestaukseen tai tuotantoon. ✓ Kehittäjän tulee täyttää ja lähettää paketoijalle toimitusseloste, josta pitää käydä ilmi sovelluksen paketoinnille ja asennukselle välttämättömät tiedot. ✓ Kehittäjät kutsuvat ensimmäisten asennusten yhteydessä myös koolle asennuspalaverin, jossa asennusprosessi käydään yksityiskohtaisesti läpi sekä selvitetään riippuvuudet taustajärjestelmiin ja yleispalveluihin. Palaveriin kutsutaan kehittäjien lisäksi paketoija ja asentajat.
Paketoija	✓ Sovellusten paketoijat tuntevat sovellusten ja komponenttien suoritusympäristöt erinomaisesti ja hallinnoivat integrointitestausympäristöä. ✓ Heidän vastuullaan on myös tarjota apua sovelluskehittäjille ajoympäristön huomioonottamisessa ja erityisesti sovelluspakettien (EAR) määrittämisessä ja leimaamisessa. ✓ Paketoija kääntää ja koostaa sovelluspaketin järjestelmätestejä ja tuotantoympäristöä varten. Paketointi tehdään vakioidussa käännösympäristössä, jonka konfiguraatio (mm. kääntäjän versio sekä kaikille sovelluksille yhteiset komponentit ja niiden versiot) on tarkasti tiedossa ja muuttuu harvoin. Näin saadaan varmistettua kaikkien sovellusten ja komponenttien paras yhteensopivuus sekä toimintavarmuus lopullisessa suoritusympäristössä (ks. tarkemmin kappale 3.1.4.3).

	Paketoija saattaa usein hyödyntää komponenttikuvauskannan tietoja paketointiprosessin yhteydessä, jos esim. käännös ei onnistukaan aivan suunnitellusti. ✓ Paketoija lähettää myös lopulliset siirto- ja asennuspyynnöt asentajille konfiguraation hallinta -välineen kautta. Ensimmäisten siirtojen ja asennusten yhteydessä paketoija välittää toimitusselosteen asentajille. Paketoija saattaa joiltain osin täydentää toimitusselostetta.
Asentaja	✓ Asentajat tuntevat erinomaisesti tuotantoympäristön ja sen rajoitukset. ✓ Heidän tehtävinään on asentaa integraatiotestin läpikäyneet sovelluspaketit ja komponentit järjestelmätestausta varten sekä huolehtia, että asennukset tehdään oikeille palvelimille ohjelmistojen ja laitteistokuormituksen näkökulmasta. ✓ Asentajien työkaluina toimivat paketoijan lähettämä toimitusseloste ja konfiguraation hallinta -väline. ✓ Asentaja kuittaa asennukset kehittäjille, paketoijalle, testaajille ja muille toimitusselosteessa mainituille osapuolille. Asennuskuittaus toimii samalla herätteenä testaajille, jotka voivat aloittaa suunnittelemansa järjestelmätestit.
Testaaja	✓ Testaaja suunnittelee järjestelmätestit sovelluksen toiminnallisten määrittelyjen pohjalta ja suorittaa tarvittavan määrän testitapauksia, jotta järjestelmän oikea toiminnallisuus voidaan todeta. ✓ Testaajat suorittavat tarvittaessa myös kuormitustestit ja poikkeustilannetestit, joilla voidaan todeta järjestelmän skaalautuvuus ruuhkatilanteissa sekä käyttäytyminen liittymien ja taustajärjestelmien ongelmien yhteydessä. ✓ Testaajat tekevät tiivistä yhteistyötä kehittäjien kanssa testitapauksia luodessaan ja asentajien kanssa kuormitustestien aikana.

Pääsuunnittelija	✓ Pääsuunnittelija tuntee hyvin yrityksen teknisen arkkitehtuurin. ✓ Pääsuunnittelija osallistuu sovellusten teknisen ratkaisun määrittelyn kriittiseen tarkasteluun jo alkuvaiheessa, jotta suunnitelmat ja toteutukset olisivat linjassa yleisen arkkitehtuurin kanssa. Lisäksi pääsuunnittelija katselmoi valmiin teknisen ratkaisun määrittelyn. ✓ Pääsuunnittelijoiden vastuulla on myös osallistua kolmannen osapuolen komponenttien verifiointiin ja karsitaan. Hän voi tarvittaessa delegoida yksityiskohtaisemman tarkastelun kunkin osa-alueen erityisasiantuntijalle, jolla on riittävä tietämys komponentin hyväksymiseksi tai hylkäämiseksi.

Sovelluskehityksen hallintavälineisiin liittyy muutamia omia erityisroolejaan, joiden vastuulle kuuluvat tehtävät esiteltään seuraavassa taulukossa [Taulukko 2].

Taulukko 2 – Sovelluskehityksen hallintavälineisiin liittyvät erityisroolit

Rooli	Tehtävät
Versiohallinta-järjestelmän vastuuhenkilö	✓ Versionhallintajärjestelmän vastuuhenkilön tulee huolehtia, että tämä hyvin tärkeä operatiivinen järjestelmä on aina käytettävissä ja että sen sisältö eheää. ✓ Vastuuhenkilö tarjoaa tukeaan paketointivälineen vastuuhenkilölle, jotta näiden välineiden välinen integraatio saadaan toimimaan luotettavasti. ✓ Tämän roolin vastuulla on myös hallinnoida järjestelmän käyttöön vaadittavia käyttöoikeuksia. ✓ Vastuuhenkilön tulee myös koordinoida versionhallintavälineen ohjelmistopäivityksiä.
Paketointi-välineen vastuuhenkilö	✓ Paketointivälineen vastuuhenkilön vastuulla on huolehtia välineen ylläpidosta ja jatkokehittämisestä jatkuvasti muuttuvan ja kehittyvän arkkitehtuurin tarpeiden mukaisesti. ✓ Tämän roolin tehtäviin kuuluu myös paketointivälineen ja

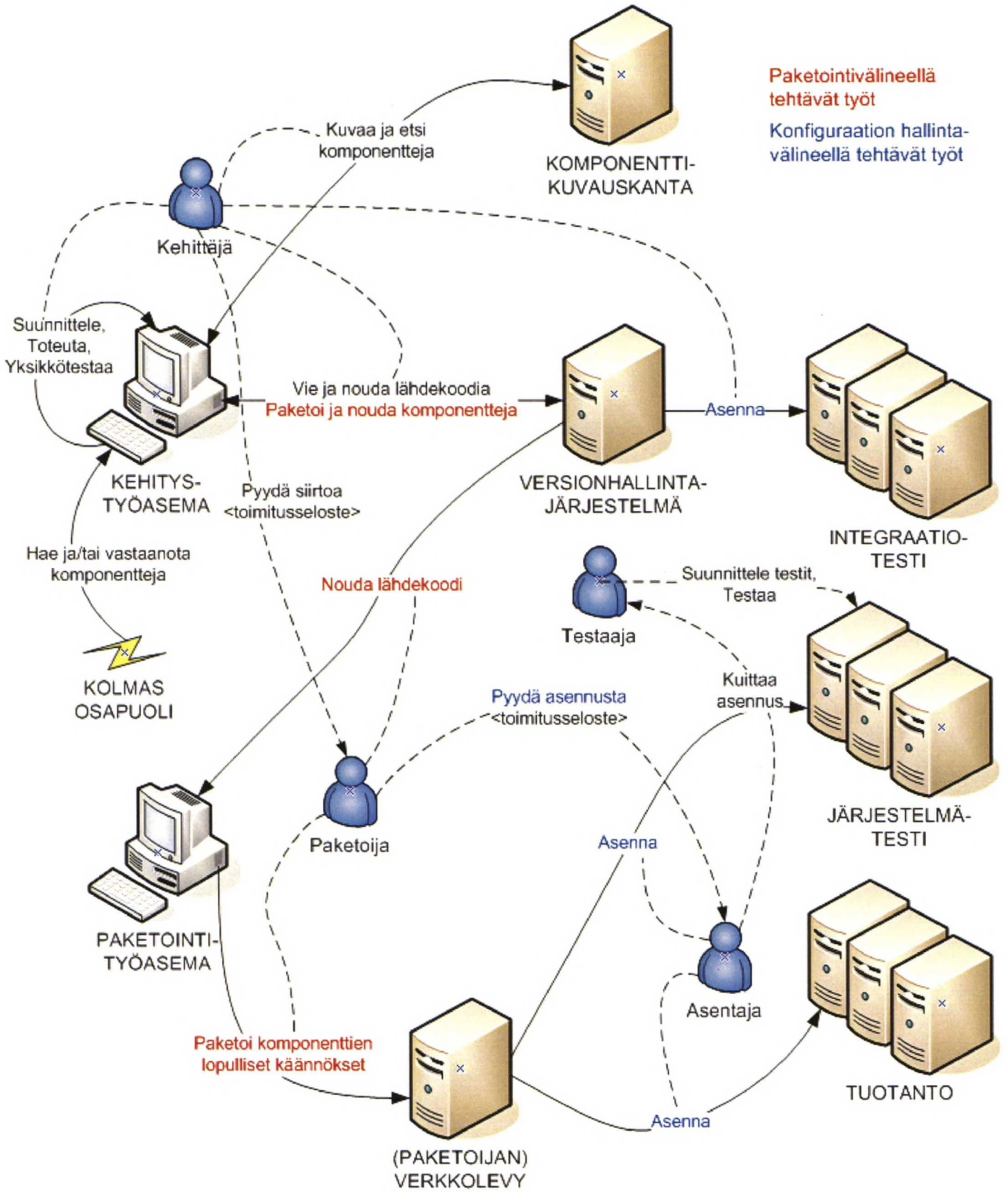
	versionhallintavälineen välisen integraation toimivuuden varmistaminen. Tässä asiassa hän tekee tiivistä yhteistyötä versionhallintajärjestelmän vastuuhenkilön kanssa. ✓ Roolin vastuulla on myös konfiguroida paketointivälineeseen liittyviä erikoisasetuksia, kuten esim. estää tiettyjen komponenttien käyttäminen tai määritellä jokin komponentti korvattavaksi tietyllä versiolla tai vaihtoehtoisella komponentilla.
Komponentti-kuvauskannan tekninen vastuuhenkilö	✓ Tämän roolin on pidettävä huolta siitä, että komponenttikuvauskanta on kaikkien käytettävissä. ✓ Paketointivälineen ja komponenttikuvauskannan välinen integraatio on myös tämän roolin vastuulla. Hän tekee sen yhteistyössä paketointivälineen vastuuhenkilön kanssa. ✓ Vastuuhenkilön tulee osallistua myös kaikkiin komponenttikuvauskantaa koskeviin kehityshankkeisiin.
Komponentti-kuvauskannan sisällöllinen vastuuhenkilö	✓ Roolin vastuulle kuuluu komponenttikuvauskannan sisällön eli sinne kuvattujen komponenttien tietojen riittävyyden tarkastelu ja toimenpiteisiin ryhtyminen puutteiden ilmetessä. ✓ Sisällöstä vastaavan henkilön vastuulle kuuluu myös hyvä arkkitehtuurin tuntemus sekä karkealla tasolla kaikkien komponenttikuvauskannassa kuvattujen komponenttien tunteminen. Hän toimii siis kehittäjien apuna, jos nämä eivät onnistu löytämään ennakoimaansa komponenttia komponenttikuvauskannasta.
Konfiguraation hallinta -välineen vastuuhenkilö	✓ Tämän roolin vastuulla on määritellä konfiguraation hallinta -välineeseen kaikki integraatiotesti-, järjestelmätesti- ja tuotantosuoritusympäristöt. ✓ Lisäksi roolin vastuulla on huolehtia, että väline on aina asentajien käytettävissä ja että se toimii halutulla tavalla. ✓ Konfiguraation hallinta -välineen vastuuhenkilö vastaa välineen ylläpidosta ja jatkokehittämisestä muuttuvan arkkitehtuurin ja suoritusympäristövaatimusten mukaisesti.

5.3 Toimintaympäristö

Seuraavan sivun kuvaan [Kuva 7] on hahmoteltu karkealla tasolla edellä kuvattujen roolien tehtäviä sekä tiedon kulku kohdeyrityksen toimintaympäristössä. Kuvasta puuttuvat sovelluskehityksen hallintavälineisiin liittyvät erityisroolit.

Sovellusten ja komponenttien toiminnan testaaminen jakaantuu neljään ympäristöön: kehittäjän työasemaan sekä integraatiotesti-, järjestelmätesti- ja tuotantoympäristöihin. Ensiksi mainitussa kehittäjä suorittaa toteuttamiensa komponenttien yksikkötestit. Integraatio- ja järjestelmätestiympäristöissä testataan komponenttien keskinäistä yhteistyötä sekä kokonaisten sovellusten tai järjestelmien toimintaa. Tuotantoympäristö on sovellusten lopullinen suorituspaikka, jossa ne ovat liiketoimintojen saatavilla. Kehittäjän työasemaa lukuun ottamatta komponenttien ja sovellusten asennuksia hallinnoidaan konfiguraation hallinta -välineellä. Asennuksen tekijä eroaa integraatiotestiympäristön sekä järjestelmätesti- ja tuotantoympäristöjen välillä. Kehittäjä asentaa komponentit ja sovellukset integraatiotestiympäristöön siinä missä järjestelmätesti- ja tuotantoympäristön asennukset tehdään asentajien toimesta.

Kuvassa on eroteltu kehitystyöasema ja paketoijan työasema toisistaan. Tähän on syynä paketoijan työaseman huolellisemmin hallittu käännös- ja paketointiympäristö. Sovelluspakettien ja komponenttien paketointi järjestelmätesti- ja tuotantokäyttöä varten tehdään nimenomaan paketoijan työasemalla. Lisäksi nämä komponentit ja asennuspaketit tallennetaan paketoijan hallinnoimalle verkkolevylle. Integraatiotestiä varten paketoidut komponentit sen sijaan tallennetaan versionhallintajärjestelmään paketointivälineen toimesta.



Kuva 7 – Sovelluskehitysprosessin toimintaympäristö

6. KOMPONENTTIKUVAUSKANNAN VAATIMUKSET

"Besides improving software reusability, component characterization also provides better understanding of architecture, better retrieval, better usage and better cataloguing."

Nasib S. Gill, Importance of Software Component Characterization for Better Software Reusability [Gil06]

6.1 Komponenteista kuvattavat tiedot

6.1.1 Komponenttien dokumentointi

Sametinger [Sam97] on koonnut kattavan tietopaketin uudelleenkäytettävien komponenttien dokumentoinnista ja sen luokittelusta. Hänen mielestään komponenttien dokumentaation tulee olla itse komponenttien tapaan itseriittoista, muunneltavaa, mukautettavaa ja laajennettavaa. Itseriittoisuus tarkoittaa, että komponentista saatavilla oleva dokumentaatio riittää sellaisenaan kullekin kohderyhmälle ja että dokumentaatioriippuvuudet pidetään minimissään.

Dokumentoinnin kohderyhmiä on kaikkiaan neljä: loppukäyttäjät, esimiehet, kehittäjät ja ylläpitäjät. Näistä lähinnä kaksi viimeistä ovat kohdeyrityksen suunnitelman mukaisia käyttäjiä. Ylläpitäjäkohderyhmään voidaan sisällyttää paketoijat ja asentajat. Sovellusten loppukäyttäjille luodaan omat käyttöohjeensa, jotka ei millään muotoa ole komponenttikohtaisia. Esimiesten kanssa samaan ryhmään sijoitetaan myös projektihallinnolliset henkilöt. Tämän kohderyhmän henkilöt hyödyntävät kehitysorganisaation asiantuntijoidensa apua suunnitellessaan ja päättäessään hankkeiden miehityksestä ja niissä uudelleenkäytettävistä tai kehitettävistä komponenteista.

Sametignerin [Sam97] mukaan uudelleenkäytettävien komponenttien dokumentoinnille on nähtävissä neljä motivaatiotekijää:

- ✓ komponenttien arviointi useiden kanditaattien joukossa,
- ✓ komponenttien toiminnallisuuden ymmärtäminen,
- ✓ komponenttien käyttö tietyssä ympäristössä ja
- ✓ komponenttien muuntaminen tiettyjen tarpeiden mukaisiksi.

Organisaation on tarpeen luoda komponenttien dokumentaatiolle tarvittavat standardit, jotta voidaan taata tietojen saatavuus, täydellisyys ja johdonmukaisuus. Tämä puoltaa lomakelähtöistä kuvaustapaa sen sijaan, että käyttäjille sallittaisiin vapaa sanallinen kuvaustapa komponenttiensa dokumentoimiseen.

Sametingerin mukaan uudelleenkäytettävistä komponenteista kuvattavat tiedot jakaantuvat seuraaviin alakategorioihin:

- ✓ Yleinen informaatio
- ✓ Uudelleenkäytön informaatio
- ✓ Hallinnollinen informaatio
- ✓ Arviointia tukeva informaatio
- ✓ Muu informaatio

Yleisen informaation perusteella käyttäjien on mahdollista rajata tarkasteltavien komponenttien joukkoa. Se ei siis riitä vielä sopivan komponentin valintaan vaan ainoastaan komponenttitarjonnan yleiskuvan muodostamiseen. Yleiseksi informaatioksi lasketaan seuraavat tiedot:

- ✓ Johdanto: Komponentin nimi ja yleiskuvaus.
- ✓ Luokitus: Komponentin luokittelemiseen tarvittavat tiedot, kuten esim. kohdealusta.
- ✓ Toiminnallisuus: Komponentin tarjoamat palvelut eli ulospäin näkyvät operaatiot
- ✓ Alustat: Alustat, joilla komponentti soveltuu käytettäväksi.
- ✓ Uudelleenkäytön tila: Jokin kvalitatiivinen arvo, joka määrittelee uudelleenkäytettävyyden esim. kehittämisen, testauksen tai ylläpidon näkökulmasta.

Uudelleenkäytön informaation avulla käyttäjät voivat ottaa tietyn komponentin käyttöönsä. Tiedot auttavat tietysti myös lopullisen komponenttivalinnan tekemisessä:

- ✓ Asennus: Ohjeet miten komponentti otetaan osaksi jotain järjestelmää.
- ✓ Rajapintojen kuvaukset: Mahdollisimman yksityiskohtainen kuvaus komponentin rajapinnoista, joiden tulee kattaa koko komponentin toiminnallisuus.
- ✓ Integrointi ja käyttö: Opastavat oikeaoppiseen uudelleenkäyttöön ja esim. kertovat miten toimia ongelmatilanteissa.
- ✓ Muokauttaminen: Ohjeet kuinka komponentti on mahdollista mukauttaa tiettyjä tarpeita varten.

Hallinnollinen informaatio auttaa käyttäjää samalla tavalla kuin uudelleenkäytön informaatio, mutta hieman eri näkökulmasta:

- ✓ Hankinta ja tuki: Yhteystietoja, joiden kautta saa apua esim. komponenttia käyttöön otettaessa tai sitä muokattaessa. Lisäksi kuvataan komponentin saatavuus ja omistajuustiedot.
- ✓ Kaupalliset ja lailliset rajoitukset: Nämä saattavat rajoittaa komponentin käyttöä ja muokkaamista. Kuvataan myös mahdollinen komponenttiin liitetty lisenssi.
- ✓ Historia ja versiot: Käsittää komponentin koko versiohistorian. Kustakin versiosta on löydyttävä avaintiedot (versioleima, päivämäärä, tekijä jne.) ja tiedot yhteensopivuudesta eri versioiden välillä.

Arviointia tukeva informaatio on tarkoitettu ennen kaikkea komponentin valintaan ja soveltuvuuden määrittämiseen liittyvien työvaiheiden avuksi. Tämän informaation avulla rajataan edelleen yleisen informaation perusteella rajattua komponenttijoukkoa:

- ✓ Määritykset: Komponentin toiminnalliset ja ei-toiminnalliset vaatimukset.
- ✓ Laatu: Jonkinlainen Kvantitatiivinen ja kvalitatiivinen arvostelu, jonka perusteella komponentin laadusta voidaan tehdä johtopäätöksiä. Tähän liittyvät mm. komponentille tehtyjen testien tulokset.
- ✓ Suorituskyky- ja resurssivaatimukset: Monipuoliset tiedot suorituskykyyn vaikuttavista tekijöistä sekä prosessoriajan, muistin ja muiden järjestelmäresurssien käyttötiedot.

- ✓ Vaihtoehtoiset komponentit: Viittaukset muihin samaan käyttötarkoitukseen soveltuviin komponentteihin.
- ✓ Tiedostetut viat ja ongelmat: Listaus mm. ratkaisemattomista ongelmista ja tavoitelluista/halutuista parannuksista.
- ✓ Rajoittuneisuus ja rajoitukset: Esim. rajautuminen tiettyyn ohjelmointikieleen, sovelluskehykseen, suoritusalustaan tai käyttöjärjestelmään.
- ✓ Mahdolliset parannukset: Tiedot mahdollisista parannuskohteista liittyen esim. uudelleenkäytettävyyteen tai ylläpidettävyyteen.
- ✓ Testaustuki: Tiedot komponentin testitapauksista, -ympäristöistä ja -penkeistä.
- ✓ Riippuvuussuhteet: Riippuvuudet muista komponenteista eli mitä muita komponentteja ko. komponentin käyttöönotto vaatii.

Muu informaatio käsittää loput uudelleenkäyttöön liittyvät komponentin tiedot, jotka eivät kuulu yhteenkään edellä esitettyyn kategoriaan:

- ✓ Järjestelmädokumentaatio: Tämä käsittää kaiken dokumentaation komponentin kehityksen alusta asti (vaatimusmäärittely, tekninen suunnitelma, toteutuksen kuvaus). Näistä on hyötyä, kun komponenttia ylläpidetään tai muokataan uusien tarpeiden mukaiseksi.
- ✓ Viitteet: Viittaukset esim. kirjallisuuteen tai muihin dokumentteihin, jotka sisältävät hyödyllistä tietoa komponentin uudelleenkäyttöä ajatellen.
- ✓ Lukemista helpottavat tiedot: Hakemisto, sisällysluettelo, kuva- ja taulukkolistat ym. tiedot, jotka helpottavat laajan dokumentaation läpikäymistä.

Uudelleenkäytettävistä komponenteista kuvattaviin tietoihin sisältyy siis sekä teknisiä että kaupallisia ja hallinnollisia asioita. Kaupallisuus on teknisen näkökulman rinnalla olennainen, koska komponentit saattavat olla niin itsenäisiä tuotteita, että niihin liittyy esim. yksittäisiä komponentteja koskevia lisenssirajoituksia tai omistussuhteita.

Kaikkia edellä mainittuja tietoja ei ole tarpeenmukaista kuvata kunkin komponentin kohdalla yksityiskohtaisesti. Tähän on useita syitä. Ensinnäkään kaikki tiedot eivät sovellu käytettäväksi kunkin komponentin tapauksessa. Lisäksi edellä kuvautut tiedot menevät osittain päällekkäin toistensa kanssa, jolloin niiden orjallisesta noudattamisesta syntyisi toisteista tietoa. Toisteisuutta tulee välttää, koska se hankaloittaa dokumentaation ylläpitoa. Pitkien yksityiskohtaisten dokumenttien

sijaan on mahdollista soveltaa minimalistista dokumentaatiota [Hol01]. Minimalistinen dokumentaatio sisältää vain osittaisia ja suoranaisesti puutteellisiakin tietoja, jolloin se edellyttää lukijaltaan tavallista korkeampaa motivaatiota tutkia itsenäisesti dokumentin kohdetta tarkemmalla tasolla. Samalla kuitenkin säästetään komponentin kuvaamiseen kuluvassa ajassa.

Dokumentaatio on jaettavissa kahteen pääkategoriaan: dokumentaatio kehittäjille eli komponentin käyttäjille sekä dokumentaatio komponentin ylläpitoa varten. Myös uudelleenkäyttöä on kahta tyyppiä: suora uudelleenkäyttö eli komponentin ottaminen osaksi järjestelmää sellaisenaan sekä ideoiden kierrätys eli esim. komponentin toteutustavan tai rajapinnan soveltaminen tai mukauttaminen uusien tarpeiden mukaiseksi. Kunkin dokumentin luonteesta selviää nopeasti mihin tarkoitukseen se on luotu tai keille se on kohdistettu. Tätä jaottelua ei siis tarvitse erikseen kuvata kunkin dokumentin kohdalla.

6.1.2 Java-komponenteista kuvattavat tiedot

Seuraavaan taulukkoon [Taulukko 3] on koottu Java-komponenttien kaikki potentiaaliset kuvailutiedot edellä esiteltyjen uudelleenkäytettävistä komponenteista kuvattavien tietojen, Hollannin määrittelemän uudelleenkäytön dokumentaation [Hol01] ja kohdeyrityksen kokemusten pohjalta. Taulukossa esitettyjä tietoja ei ole tarkoitettu käytettäväksi sellaisenaan, erillisinä ja yksittäisinä syötteinä, komponenttikuvauskannan toteutuksessa. Sen sijaan kuvailutietoja kannattaa niputtaa selkeisiin osakokonaisuuksiin, jolloin tietojen hahmottaminen, ylläpito ja esittäminen on yksinkertaisempaa ja nopeampaa.

Taulukko 3 – Java-komponenteista kuvattavissa olevat tiedot

#	Vaatimus	Kuvaus
1.	komponentin nimi	*Yksilöivä* komponentin nimi
2.	Lyhyt kuvaus	Komponentti kuvattuna ytimekkäästi noin yhdellä virkkeellä
3.	Yleiskuvaus	Karkean tason sanallinen kuvaus siitä, mitä toiminnallisuuksia komponentti tarjoaa ja mitä sen avulla on tarkoitus tehdä.

#	Vaatimus	Kuvaus
4.	Avainsanat	Joukko termejä, jotka voidaan mieltää synonyymeiksi komponentin nimelle tai sen toteuttamille toiminnoille. Näitä voidaan hyödyntää mm. hakutoiminnoissa.
5.	Komponentin tyyppi	Mahdollisia Java-komponentin tyyppejä ovat JAR (resurssi), EJB (palvelu), WAR (web-sovellus) ja RAR (resurssiadapteri).
6.	Vastuuhenkilöt	Tiedot henkilöistä, joilla on vastuu komponentin ylläpidosta ja käyttötuesta.
7.	Ohjeet ja käyttöesimerkit	Komponentin käyttöönotossa auttavat ohjeet ja käyttöesimerkit.
8.	Rajapintakuvaukset (Javadoc)	Java-komponentin rajapinta on kuvattu lähdekoodin yhteydessä Javadoc-kommentein, joista myöhemmin generoidaan Javadoc-dokumentaatio. Javadoc-dokumentaatio voidaan konfiguroida kattamaan myös komponentin sisäiset rajapinnat (private ja protected -määreiden takana olevat muuttujat ja metodit).
9.	Luokittelu	Komponentin luokittelu on useimpia muita vaatimuksia monimutkaisempi sekä riippuvaisempi yrityksen arkkitehtuurista. Luokittelu voidaan tehdä useasta eri näkökulmasta: esim. toteutuspatternien, monitasoarkkitehtuurin, liiketoiminnan tai sovelluskehyksen näkökulmasta. [Crn02]
10.	Soveltuvat suoritusympäristöt	Komponentti voidaan luokitella myös suoritusympäristön perusteella. Yrityksellä voi olla useita erilaisia Java-suoritusympäristöjä. Suoritusympäristöinä toimivat eri valmistajien sovelluspalvelimet tai erilliset Java-virtuaalikoneet. Lisäksi yritys on voinut jakaa ympäristöt sovellusalustojen tai -kehysten mukaan esim. tavanomaisempaan palvelukehitykseen (Java EE) ja kanavakerrokseen keskittyvään portaalikehitykseen (mm. BEA WebLogic Portal [BEAWP]).

#	Vaatimus	Kuvaus
11.	Asennus- ja konfigurointiohjeet	Kuvaus miten komponentti otetaan käyttöön ja mitä konfigurointia se mahdollisesti vaatii esim. suoritusympäristön tai käyttöjärjestelmän osalta.
12.	Uudelleenkäytön tila	Tällä vesileimataan komponentin tila sen uudelleenkäyttäjien eli kehittäjien näkökulmasta. Tilaan voidaan kytkeä myös aikataulutietoa (esim. milloin komponentti on valmistumassa).
13.	Vaihtoehtoiset komponentit	Viittaukset vastaavaa toiminnallisuutta sisältäviin komponentteihin, joita käyttäjän olisi syytä tarkastella jos ko. komponentti ei syystä tai toisesta sovellu hänen tarpeisiinsa.
14.	Web Service -tiedot	Jos komponentista generoidaan Web Service [JWS], on komponenttikuvauskannassa mahdollista kertoa generoidun Web Servicen avaintiedot. Web Service-generoinnin yhteydessä syntyy alkuperäiselle komponentille rinnakkainen komponentti, johon ei kuitenkaan liity omaa lähdekoodiaan vaan ainoastaan Web Service-rajapinnan kuvailevat resurssitiedostot (lähinnä asennuskuvaimet).
15.	Kaupalliset ja lailliset rajoitukset	Komponentin mahdollinen käyttölisenssi ja muut ehdot. Lisenssitieto koskee ensisijaisesti kolmannen osapuolen komponentteja. Itse toteutettuihin komponentteihin liittyy harvoin mitään lisenssiä, paitsi jos ne on tarkoitettu myös yrityksen ulkopuolelle myytäväksi.
16.	Suorituskyky- ja resurssivaatimukset	Tiedot komponentin suorituskykyyn vaikuttavista tekijöistä sekä prosessoriajan, muistin ja muiden järjestelmäresurssien käyttöaste.
17.	Tiedostetut viat ja ongelmat	Lista mm. ratkaisemattomista ongelmista ja tavoitelluista tai halutuista parannuksista.

#	Vaatimus	Kuvaus
18.	Tulossa olevat muutokset ja parannukset	Tiedot mahdollisista parannuskohteista liittyen esim. uudelleenkäytettävyyteen tai ylläpidettävyyteen. Mahdollisesta komponentin tulevasta ja/tai käynnissä olevasta kehitystyöstä on siis syytä mainita tässä yhteydessä.
19.	Testaustuki	Tiedot komponentin testitapauksista, -ympäristöistä ja -penkeistä.
20.	Sijainti versionhallinnassa	Komponentin lähdekoodien ja muiden resurssitiedostojen sijainti versionhallintajärjestelmässä.
21.	Historia ja versiot	Käsittää komponentin koko versiohistorian. Kustakin versiosta kerrotaan yksiselitteinen versiolabel, luontiaika, tekijä, vapaamuotoinen selite. Selitteessä on syytä painottaa versioiden yhteensopivuutta muiden versioiden kanssa. Myös komponentin kuvailutietoihin tehdyistä muutoksista voi pitää kirjaa. Näiden yhteydessä on hyv kirjata vähintään ajankohta, tekijä ja muutoksen luonne.
22.	Riippuvuudet	Riippuvuudet voivat olla saatavilla kahteen suuntaan: komponentin riippuvuudet muista komponenteista sekä muiden komponenttien riippuvuudet kuvatusta komponentista. Käänteiseen riippuvuushakutoiminnallisuuteen vaikuttavat eritoten riippuvuuksien tallennustapa, komponenttien kokonaismäärä sekä käytettävissä olevat laskentaresurssit. Tämä toiminnallisuus onkin syytä määritellä hyvin harkiten yrityksen sisäisten tarpeiden mukaisesti. Yksi suurimmista tehokkuuteen vaikuttavista päätöksistä on vastaus kysymykseen ”uloitetaanko haku komponenttien kaikkiin versioihin vai esim. vain tuoreimpiin versioihin?”.

#	Vaatimus	Kuvaus
23.	Hinta	Komponentin hankinta- ja/tai myyntihinta sekä mahdollinen kurantti markkinahinta, jos komponentti on ehtinyt ikääntyä. Myös komponentin käyttöön voi liittyä oma hinnoittelunsa.
24.	Käyttöaste	Kvalitatiivinen tai kvantitatiivinen arvio siitä kuinka laajalti komponentti on käytössä yrityksen järjestelmissä.
25.	Omistaja	Komponentin omistavan tahon tiedot.
26.	Kokemusten vaihtoa	Käyttäjille voi tarjota mahdollisuuden vaihtaa keskenään kokemuksia komponentin käytöstä. Toteutus on mahdollista esim. jonkin Wiki-tuotteen [Wiki] avulla.

6.2 Toiminnallisuus

6.2.1 Haku

Ehkä tärkein komponenttikuvauskannan toiminnallinen vaatimus on monipuolinen hakutoiminto. Haku tulee voida kohdistaa haluttuihin kuvailutietoihin. Lisäksi hakutulosta tulee olla mahdollista rajata (esim. luokittelutietojen perusteella). Tämä mahdollistaa monenlaisia käyttötapoja, joista alla listattuna vain murto-osa esimerkinomaisesti:

- ✓ komponenttien haku tiettyyn käyttötarkoitukseen avaintermin avulla. Esim. ”osake”, ”kurssi” tai ”päivämäärä”.
- ✓ haku komponentin nimen perusteella esim. selvittääkseen onko jokin tietty nimeltä tunnettu kolmannen osapuolen komponentti jo hankittuna.
- ✓ haku vastuuhenkilön nimen perusteella esim. hakeakseen kaikki omalla vastuullaan olevat komponentit.
- ✓ mahdollisuus ottaa hakutulokseen mukaan myös käytöstä poistetut komponentit.
- ✓ käyttäjä voi kohdistaa haun pelkästään JavaDoc-dokumentaatioon, jos haluaa etsiä mihin komponenttiin tai komponentteihin jokin tietty rajapinta tai luokka sisältyy. Tästä on apua esim. java.lang.ClassNotFoundException -tyyppisten käännösongelmien selvittämisessä.

Hakuun liittyy kuitenkin muutama merkittävä haaste. Kehittäjien sanasto on harvoin kauttaaltaan yhteneväinen, minkä vuoksi hauissa saatetaan käyttää komponentin kuvaamiseen käytetyistä termeistä poikkeavia hakusanoja, jolloin osa hakijan tarpeisiin vastaavista komponenteista saattaa jäädä pois hakutuloksesta. Tässä yhteydessä voisi olla apua avainsanoista, joiksi kehittäjät voisivat komponenttia kuvatessaan luetella useita eri synonyymejä. Toinen vaihtoehto on liittää haun tueksi jonkinlainen synonyymisanasto, mutta sellaisen hankkiminen voi olla hankalaa tai sen toiminta epäluotettavaa – varsinkin jollain muulla kielellä kuin englannilla. Voi myös olla, ettei moista synonyymisanastoa ole halutulle kielelle edes saatavilla. Hakuun täsmänneistä komponenteista on syytä näyttää vain kaikkein olennaisimmat tiedot tiivistetyssä muodossa (esim. yksilöivä nimi, vastuuhenkilö, lyhyt kuvaus jne.)

6.2.2 Selailu

Hakutoiminto tuskin yksinään riittää takaamaan, että käyttäjät löytäisivät aina etsimänsä komponentin. Tämän vuoksi komponenttikuvauskannasta tulisi löytyä selailutoiminto, jonka kautta käyttäjän on mahdollista nähdä haluamansa otanta kaikista komponenteista. Tämä voi perustua esim. aakkosjärjestyksessä esitettyyn komponenttilistaukseen tietyltä kirjainväliltä. Lisäksi käyttäjälle voi tarjota mahdollisuuden rajata selailunäkymässä näytettäviä komponentteja. Rajaus voi perustua kaikkiin niihin kuvailutietoihin, joiden arvojoukko on jotenkin rajattu (valintalistat ym.). Vapaasanaiseen rajaamiseen käyttäjä voi käyttää aiemmin kuvattua hakutoimintoa. Selailunäkymässä näytettävät komponenttien tiedot on syytä valita samoin periaattein kuin hakutuloksen tapauksessakin.

6.2.3 Komponenttien lisääminen

Käytännössä komponentti on syytä lisätä kuvauskantaan viimeistään, kun komponentti ensimmäisen kerran paketoidaan paketointivälineellä. On myös hyödyllistä tarjota mahdollisuus komponentin lisäämiseen komponenttikuvauskantaan ennen paketointia. Suunnitteilla olevan tai keskeneräisen komponentin lisääminen komponenttikuvauskantaan on yksi tapa tiedottaa muille kehittäjille ko. komponentin olemassa olosta. Kehittäjät saattavat tällöin löytää ko. komponentin tiedot jo hyvissä ajoin esim. suunnitellessaan uutta sovellusta tai

yleispalvelua. Etukäteen tehty kuvaus ei mene myöskään hukkaan, koska nämä tiedot voidaan noutaa pohjaksi, kun komponenttia ensimmäistä kertaa paketoidaan. Samalla käsin tehtävä lisääminen mahdollistaa sellaistenkin komponenttien kuvaamisen, joita ei syystä tai toisesta paketoida millään paketointivälineellä.

6.2.4 Komponenttien päivittäminen

Kertaalleen lisätyn komponentin tietoja on usein tarve ylläpitää myöhemmin. Tietoja on syytä päivittää etenkin uusien versioiden paketoinnin yhteydessä tai kun kuvauksessa havaitaan virheitä tai puutteita. Paketoinnin yhteydessä komponentin uusien versioiden tietojen on syytä siirtää automaattisesti komponenttikuvauskantaan. Samassa yhteydessä on hyvä voida päivittää myös muita kuvailutietoja. Tämä vaatii paketointivälineen komponenttikuvauskannan integroinnin, jotta kehittäminen oikeasti tehostuisi. Ilman integraatiota kehittäjien pitäisi käyttää eri välinettä komponenttien paketointiin ja paketoitujen komponenttien kuvaamiseen.

6.2.5 Komponenttien poistaminen

Komponentin poistaminen ei vaadi komponentin kuvailutietojen poistamista komponenttikuvauskannasta. Komponentti voidaan ainoastaan merkitä poistetuksi, jolloin se ei enää oletusarvoisesti näy hakutuloksissa tai selailunäkymässä. Yrityksessä on hyvä linjata onko kahden samannimisen komponentin olemassaolo sallittua vai pitääkö nimen olla aina yksilöllinen. Yksilöllisyysvaatimus johtaa vääjäämättä siihen, että komponentin tietoja ei voida hävittää, jotta tämäntyyppistä päällekkäisyyttä ei pääsisi syntymään. Todennäköisesti yritykset todellisuudessa aina vaativat komponenttien nimen yksilöllisyyttä ja siksi aiemmin kappaleessa 6.1.2 nimi mainittiinkin yksilölliseksi.

6.2.6 Päivitysloki

Tiedottamista voidaan helpottaa myös erityisellä näkymällä, joka olisi haluttu otanta viimeisimmistä komponenttien kuvailutietoihin tehdyistä päivityksistä. Näytettävät komponentit ja tiedot voidaan valita yrityksen tarpeiden mukaisesti. Voidaan näyttää

esim. viimeisimmät yleiskomponenttien päivitykset siten, että listalla on korkeintaan yksi merkintä komponenttia kohden.

6.2.7 Komponentin kuvailutietojen esittäminen

Komponenttikuvauskannan sisältö kulminoituu sivuun, jolla esitetään kaikki kuvailutiedot yhdestä komponentista. Tämä näkymä onkin syytä suunnitella huolella, jotta siitä saataisiin suurin hyöty. Tietojen olisi syytä olla helposti tulostettavassa muodossa. Vaikka nykyään kovasti puhutaankin paperittomasta toimistosta, tosiasia on, että ihmiset ottavat tietoja mukaan paperilla esim. palavereihin. Lisäksi tietojen tulisi olla tekstimuodossa, jotta tietoja olisi mahdollisimman helppo kopioda vaikkapa sähköpostiviestiin. Pitkiä luetteloja, kuten esim. versiohistoriaa on järkevää rajoittaa, jotta sivu olisi tiiviimpi mm. tulostamista ajatellen. Versioista voidaan esim. tarjota kaksi taulukkonäkymää: viisi (5) uusinta versiota tai kaikki versiot.

6.2.8 Muutosten vaikutusalueen kartoitus (Impact Analysis / Change Management)

> *"By managing asset and application dependencies through a well-managed repository, you're working smarter, and you're further improving the long-term benefits of software reuse for your organization. But if your reuse department doesn't have a repository, or if there is no organized, managed reuse department, then any change to a shared resource is like a shot in the dark. If that's the case, be afraid. Be very afraid."*
>
> *Anthony Meyer, Software Reuse: Why You Need a Well Managed Repository [Mey02]*

Meyerin [Mey02] mukaan muutosten vaikutusalueen kartoitus on kriittinen avaintekijä yrityksen uudelleenkäyttöhankkeiden onnistumisessa. Hänen mukaansa ilman kartoitusta uudelleenkäytöstä voi olla jopa enemmän haittaa kuin hyötyä. Jos laajalti käytössä olevaan komponenttiin tehdään muutos ilman, että on mahdollista etukäteen kartoittaa mihin muutos vaikuttaa, ajaudutaan merkittäviin ongelmiin.

Kartoittamisen ensimmäinen askel on hyödyntää komponenttien käännöksen ja suorituksen aikaisia riippuvuustietoja. Kartoitus tuottaa huomattavasti tarkemman

tuloksen, jos vaikutusta on mahdollista analysoida lähdekoodin tasolla. Tällöin saataisiin selvitettyä komponentit, jotka käyttävät oikeasti niitä rajapintoja, joihin ollaan tekemässä muutoksia. Toisaalta muutos saattaa olla komponentin sisäinen, jolloin tarkastelun alle on syytä ottaa kaikki komponentit, jotka vähänkään riippuvat muutettavasta komponentista. Tämä lista koostunee yleensä suurimmalta osin sovelluskohtaisista komponenteista, jotka hyödyntävät yleiskäyttöisiä komponentteja. Muutosten vaikutusalueen kartoitus onkin yksi tärkeimmistä toiminnallisuuksista, jonka perusteella komponenttikuvauskantaan tulisi kuvata kaikki yrityksessä käytössä olevan komponentit eikä pelkästään uudelleenkäytettäviä komponentteja.

Cerulo [Cer06] on tutkinut väitöskirjaansa varten muutosten vaikutusalueen kartoitusta monelta kantilta. Hän on löytänyt useita lähteitä, joiden mukaan asiantuntijoiden arviot vaikutusalueesta ovat useimmiten virheellisiä. Monet ovat todenneet lähdekoodin perusteella tehdyn kartoituksen olevan niin kallista, että sen käyttöönotto on mahdotonta. Onkin erittäin suuri riski lähteä toteuttaa yrityksessä Impact Analysis -toiminnallisuutta itsenäisesti – varsinkin, jos markkinoilta löytyy yksikin tähän tarkoitukseen soveltuva tuote. Kartoituksen toteuttaminen oikein ja tehokkaasti ei ole ollenkaan triviaalia, vaikka haku rajattaisiin tiettyyn ohjelmointikieleen.

Teollisuudesta löytyi vain harvoja tuotevaihtoehtoja (esim. IBM Rational ClearCase Change Management Solution [IRCCM], Borland StarTeam [BST]). Samoin akatemisesta maailmasta löytyi vain muutama tuote (esim. Chianti [Ren04, Ren05], Jinpa [Can06]). Aihealue on erittäin haastava ja sitä onkin tähän päivään asti käsitelty enemmän akateemisessa maailmassa kuin yritysten toimesta. Toiminnallisuus on yleensä upotettu osaksi laajempia SCM-tuotekokonaisuuksia tai versionhallintajärjestelmiä. Tämä nostaa muutosten vaikutusalueen kartoittamisen hintaa, jos yritykseen on jo hankittu samaa toiminnallisuutta sisältäviä välineitä muilta toimittajilta.

6.3 Liittymät

6.3.1 Paketointiväline

Paketointiväline on osa operatiivista prosessia, joten sen toiminta ei saa häiriintyä, vaikka komponenttikuvauskannan palvelut eivät olisi saatavilla. Paketoinnin yhteydessä siirtämättä jääneet tiedot voidaan siirtää komponenttikuvauskantaan, kun sen palvelut ovat jälleen saatavilla.

Paketointivälineen tulee myös tarkistaa, että kullakin luotavalla komponentilla on yksilöivä nimi. Lisäksi komponenttien väliset riippuvuudet kuvataan paketoinnin yhteydessä tehtävää käännöstä ja paketin koostamista varten, joten niitä ei ole järkevää kuvata toiseen kertaan komponenttikuvauskannan kautta, vaan tämä tieto on saatavilla joko paketointivälineen kautta. Riippuvuustiedot voidaan myös toimittaa komponenttikuvauskantaan, mutta niitä tulisi voida ylläpitää ainoastaan paketointivälineellä.

Paketointivälineen yhteydessä on hyödyllistä tarjota lähestulkoon sama kuvailutoiminnallisuus kuin erillisen komponenttikuvauskantasovelluksenkin kautta. Tämä on tehokasta, koska kehittäjä voi tällöin kunkin paketoinnin yhteydessä helposti tarkistaa kuvailutiedot ja päivittää niitä tarvittaessa.

Kuvailutietojen siirto on järkevää jakaa osiin. Yksi vaihtoehto on jakaa siirto kahteen osaan: versiokohtaiset tiedot ja muut kuvailutiedot. Tällöin aina uuden version paketoinnin yhteydessä tehdään yhdestä kahteen (1-2) siirtotoimenpidettä. Aina toimitetaan uuden version tiedot, mutta muut kuvailutiedot voi jättää siirtämättä, jos käyttäjä ei ole päivittänyt niitä.

6.3.2 Versionhallinta

Komponenttikuvauskantaa ja versionhallintajärjestelmää ei kannata integroida ajantasaliittymänä. Yrityksessä tulee kuitenkin tarve täsmätä komponenttikuvauskannan versiotiedot versionhallintajärjestelmästä löytyvien tietojen kanssa. Tähän tarkoitukseen voidaan luoda offline-liittymä, kuten esim. eräajosovellus, joka suorittaa tietojen täsmäyksen pyydettäessä. Tämä on tarpeellista,

sillä ei voida taata komponenttikuvauskannan esteetöntä saatavuutta paketointivälineelle kaikissa tilanteissa.

6.3.3 Konfiguraation hallinta -väline

Konfiguraation hallinta -välineen tiedossa on komponenttien todellinen käyttö eri suoritusympäristöissä. Jos komponenttikuvauskantaan halutaan ajanmukaisia tietoja esim. komponentin käyttöasteesta tai niiden todellisista suorituksen aikaisista konfiguraatioista, on konfiguraation hallinta -väline tai sen tietovarastot integroitava komponenttikuvauskantaan. Koska konfiguraation hallinta -väline on paketointivälineen tapaan operatiivinen sovellus, ei sen toiminnallisuus saa häiriintyä, vaikka komponenttikuvauskannan palvelut eivät olisi käytettävissä.

7. KOMPONENTTIKUVAUSKANNAN TOTEUTUS

7.1 Komponenteista kuvattavat tiedot

Seuraavassa taulukossa [Taulukko 4] esitellään kaikki kohdeyrityksen komponenttikuvauskannan toteutukseen mukaan otetut kuvailutiedot.

Taulukko 4 – Komponenteista kuvattavat tiedot kohdeyrityksen toteutuksessa

#	Vaatimus	Kuvaus
1.	komponentin nimi (+ tyyppi)	*Yksilöivä* komponentin nimi, joka sisältää tiedon komponentin tyypistä. Nimi on muotoa <nimi>_<tyyppi>. Esim. *tuotehallinta_ejb* tai *xmltools_jar*
2.	Kuvaava nimi	Komponentti kuvattuna ytimekkäästi noin yhdellä virkkeellä.
3.	Yleiskuvaus	Karkean tason sanallinen kuvaus siitä, mitä toiminnallisuuksia komponentti tarjoaa ja mitä sen avulla on tarkoitus tehdä. Yksinkertainen tekstin muotoileminen mahdollisia: ✓ Lihavoitu teksti (HTML: <B>) ✓ Tasamittainen teksti koodiesimerkeille (HTML: <CODE>).
4.	Vastuuhenkilöt	Käyttäjätunnukset yhdestä kolmeen (1-3) henkilöltä, joilla on vastuu komponentin ylläpidosta ja käyttötuesta.
5.	Ohjeet, käyttöesimerkit, vaatimukset, rajoitukset ym.	Komponentin käyttöönotossa auttavat ohjeet, käyttöesimerkit, vaatimukset, rajoitukset ym. Samat muotoilut ovat mahdollisia kuin yleiskuvauksen yhteydessä.

#	Vaatimus	Kuvaus
6.	Rajapintakuvaukset (Javadoc)	Paketointiväline generoi komponentin paketoinnin yhteydessä automaattisesti Javadoc-rajapintadokumentaation, joka toimitetaan komponenttikuvauskantaan. Tämä edellyttää, että komponentti paketoidaan lähdekoodista. Lähdekoodittomien eli valmiiksi käännettyjen komponenttien paketoinnin tapauksessa Javadoc-dokumentaatio on toimittetavissa samaan tapaan, jos se on saatavilla. Javadoc on haluttaessa mahdollista päivittää myös myöhemmin erillään paketoinnista.
7.	Luokittelu	Kohdeyrityksessä valittiin karkea liiketoiminta- ja arkkitehtuurilähtöiset luokittelut yhdistävä luokittelutapa. Luokittelu on kolmitasoinen: ✓ Tyyppi: sovellus / yleinen ✓ Kerros: kanava / palvelu ✓ Kohde: intranet / Internet ✓ Lähdekoodi kohdeyrityksen omistuksessa: kyllä / ei Kanavakerros-komponentti on kohdeyrityksen käyttämä nimitys käyttöliittymä- tai esitystapakerroksen palveluja toteuttavasta tai tarjoavasta komponentista. Kohde-luokittelukriteeri on samalla sekä liiketoiminnallinen että arkkitehtuurinen luokittelutekijä. Esim. liiketoiminnan näkökulmasta komponentti on toiminnallisuudeltaan sekä liittymiltään todennäköisesti hyvin erilainen, jos se soveltuu yrityksen sisäisten järjestelmien (intranet) toteutukseen, mutta ei julkiseen asiakaskanavaan (Internet). Luokittelun yhteyteen on myös mahdollista syöttää vapaamuotoista lisätietoa.

#	Vaatimus	Kuvaus
8.	Tila	Valittavissa olevat tilat: ✓ kehityksessä ✓ saa käyttää ✓ ei saa käyttää ✓ poistettu. Tilan yhteyteen on myös mahdollista syöttää vapaamuotoista lisätietoa esim. valmistumisaikataulusta.
9.	Web Service -tiedot	Paketointivälineellä voidaan komponentista määritellä generoitavan Web Service. Kustakin komponenteista, josta on kytketty Web Service-generointi päälle, välitetään seuraavat avaintiedot komponenttikuvauskantaan: ✓ Web Service -nimi ✓ Web Service -paketin nimi ✓ Web Service -asiakaspaketin (client) nimi
10.	Lisenssi	Kunkin komponentin kohdalla on mahdollista määritellä sitä koskeva lisenssi. Valittavana on vapaan lähdekoodin yhteisöjen lisenssejä ja yritysten tekijänoikeuksia (copyright). Myös lisenssin yhteyteen on mahdollista syöttää vapaamuotoista lisätietoa.

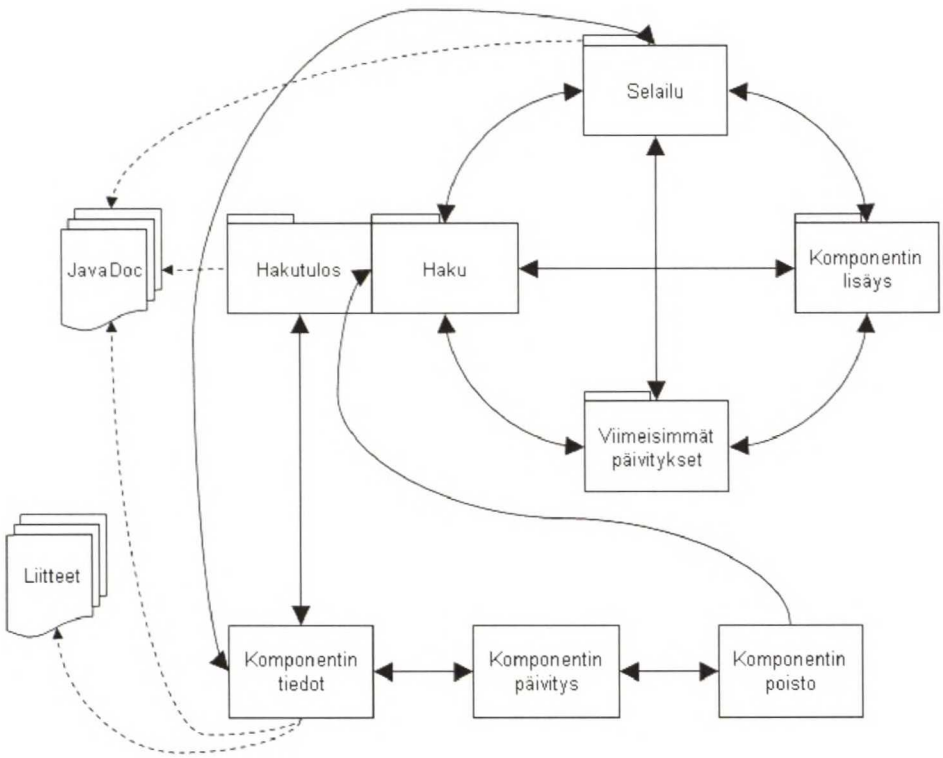
#	Vaatimus	Kuvaus
11.	Historia ja versiot	Kustakin komponentista on nähtävillä sen koko muutoshistoria ja kaikki versiot. Kuhunkin muutoshistoriamerkintään liittyy: ✓ Muutoksen ajankohta ✓ Muutoksen tekijä ✓ Muutostyyppi (Lisäys/päivitys/paketointi/poisto) ✓ Muutosselite Kustakin versiosta kerrotaan: ✓ Paketointiajankohta ✓ Paketoija (tekijä) ✓ Versioleima (versionhallinta-label) ✓ Version kuvaus (selite)
12.	Liitetiedostot	Kunkin komponentin kohdalle on mahdollista tallentaa liitetiedostoja, joiden tiedostotyyppi ja sisältö on komponentin kuvaajan päätettävissä. Liitteet voivat olla esim. kuvia, yrityksen ulkopuolisten tahojen toimittamia dokumentteja, määrittelyjä, sovelluskirjoja, asennusohjeita jne. Yksi liitteiden vahvuus on mahdollisuus hyödyntää kunkin dokumenttityypin tarjoamia muotoiluja, jotka selainkäyttöliittymän kautta syötettynä tai esitettyinä olisivat työläitä tai jopa mahdottomia.

Toteutukseen valittu kuvailutietojoukko on huomattavasti kappaleessa 6.1.2 esiteltyä joukkoa pienempi. Tämä johtuu mm. siitä, että useita kuvailutietoja on yhdistelty. Esim. yleiskuvaukseen, ohjeisiin ja liitteissä on ajateltu kuvattavan mm. avainsanat, soveltuvat suoritusympäristöt, asennus- ja konfigurointiohjeet, vaihtoehtoiset komponentit, resurssi- ja suorituskykyvaatimukset, tiedostetut viat ja ongelmat, tulossa olevat muutokset ja parannukset ja testaustuki. Näiden lisäksi osa muista kuvailutiedoista on saatavilla paketointi ja konfiguraation hallinta -välineiden kautta.

Muutosten vaikutusalueen kartoitusta (Impact Analysis, kappale 6.2.8) harkittiin pitkään kohdeyrityksessä, mutta sitä ei toistaiseksi otettu mukaan sovelluskehityksen hallintavälineiden toteutuksiin.

7.2 Käyttöliittymän toiminnallisuus

Komponenttikuvauskannan selainkäyttöliittymästä löytyy kullekin toiminnolle oma sivunsa. Sivujen välinen keskustelukaavio on nähtävillä kuvassa alempana [Kuva 8]. Kuvassa haku-, selailu-, komponentin lisäys- ja viimeisimmät päivitykset -sivut ovat kukin omalla välilehdellään, joten näiden sivujen välillä siirtyminen on nopeaa. Myös hakutulos -sivu näytetään haku -välilehden alla. Komponentin tiedot, komponentin päivitys ja komponentin poisto -sivuilla ei ole välilehteä ja niiltä voi siirtyä toisille sivuille vain sivulla kulloinkin näkyvien toimintojen kautta (esim. paluu ja paluu hakuun -linkit). Kultakin sivulta on mahdollista palata sovelluksen alkuun (haku), vaikka tätä ei olekaan keskustelukaavioon piirretty.



Kuva 8 – Selainkäyttöliittymän keskustelukaavio

7.2.1 Haku

Haku-sivun tärkeimmät kohdat ovat hakusanat -kenttä ja Hae -painike. Näiden lisäksi käyttäjä voi valita mihin kaikkiin tietoihin haku ulotetaan ja millä tavalla hakutulosta rajataan.

Haun voi yksitellen (checkbox) ulottaa kuhunkin kuvailutietoon pois lukien liitteet. Näistä viisi (5) ensimmäistä on oletuksena valittuna:

- ✓ Komponentin nimi
- ✓ Kuvaava nimi
- ✓ Yleiskuvaus
- ✓ Henkilöiden nimet
- ✓ Ohjeet, käyttöesimerkit, vaatimukset/rajoitukset ym.
- ✓ Luokittelu -lisätiedot
- ✓ Lisenssi -lisätiedot
- ✓ Javadoc
- ✓ Muutoshistoria
- ✓ Tila -lisätiedot
- ✓ Web Service -tiedot

Hakutulosta voi rajata seuraavien valintojen avulla:

- ✓ Tyyppi: Sovellus / Yleinen (2 x checkbox)
- ✓ Kerros: Kanava / Palvelu (2 x checkbox)
- ✓ Kohde: Sisäinen / Ulkoinen (2 x checkbox)
- ✓ Lähdekoodi kohdeyrityksen omistuksessa: Kyllä / Ei (2 x checkbox)
- ✓ Lisenssi (valittavissa yksi lisenssi) (1 x select-valintalista)
- ✓ Tila: Kehityksessä / Saa käyttää / Ei saa käyttää / Poistettu (4 x checkbox)

Poistetuksi merkittyjä komponentteja ei oletusarvoisesti ole mukana haussa. Myöskään lisenssirajoitusta ei ole oletusarvoisesti valittuna.

Hakutulos -sivulla näytetään tiivis luettelo, jossa kustakin komponentista esitetään nimi, 1. vastuuhenkilö, kuvaava nimi sekä linkki Javadoc-dokumentaatioon, jos se on saatavilla.

7.2.2 Selailu

Selailuntoiminnon kautta on nähtävissä kaikki kuvatut komponentit. Näkymä on kuitenkin pilkottu komponentin nimen alkukirjaimen perusteella. Komponentit esitetään aakkosjärjestyksessä väleiltä A–E, F–J, K–O, P–T ja U–Z. Komponentteja on aluksi satoja ja tulevaisuudessa mahdollisesti tuhansia, minkä vuoksi niitä on mahdollista rajata, kuten haunkin yhteydessä.

Selailunäkymän rajoitusvalinnat eroavat haku-toiminnon rajoitusvalinnoista osittain:

- ✓ Komponentin tyypin (JAR/EJB/WAR/RAR) rajoittaminen mahdollista
- ✓ Lisenssi-rajoitusta ei ole mukana

Selailunäkymässä esitetään komponenteista samat tiedot kuin hakutulos -sivulla.

7.2.3 Komponentin lisäys

Komponentti on mahdollista lisätä kuvauskantaan myös selaimen kautta ennen ensimmäistäkään paketointia. Tämä mahdollistaa myös syystä tai toisesta paketoinnin ulkopuolelle jäävien komponenttien kuvaamisen yhteisellä välineellä. Tällä tavalla lisätyistä komponenteista syntyy versio- ja Web Service-tietoja vasta paketoinin kautta.

Seuraavia tietoja on mahdollista syöttää komponentin lisäys -sivulla. Tähdellä merkityt ovat pakollisia tietoja:

- ✓ Komponentin nimi*
- ✓ Komponentin tyyppi*: JAR / EJB / WAR / RAR
- ✓ Kuvaava nimi*
- ✓ Yleiskuvaus*
- ✓ Vastuuhenkilöt*: 1-3 käyttäjätunnusta
- ✓ Ohjeet, käyttöesimerkit, vaatimukset, rajoitukset ym.
- ✓ Luokitus*:
    - o Tyyppi: Sovellus / Yleinen (2 x radiobutton)
    - o Kerros: Kanava / Palvelu (2 x checkbox)
    - o Kohde: intranet / Internet (2 x checkbox)
    - o Lähdekoodi kohdeyrityksen omistuksessa: Kyllä / Ei (2 x radiobutton)

- Lisätiedot

✓ Lisenssi
- Lisenssi (1 x select-valintalista)
- Lisätiedot

✓ Komponentin tila*
- Tila: Kehityksessä / Saa käyttää / Ei saa käyttää (1 x select-valintalista)
- Lisätiedot

✓ Javadoc (ZIP-tiedoston siirto HTTP:n yli)

✓ Liitteet
- Liitteen kuvaus
- Liitetiedosto (tiedoston siirto HTTP:n yli)

Liitteet pitää syöttää yksi kerrallaan, mutta niiden määrää ei ole rajoitettu.

7.2.4 Komponentin päivitys

Komponentin päivittäminen ei juuri eroa komponentin lisäyksestä, vaikkakaan komponentin nimeä ei ole mahdollista muuttaa. Päivityksessä käyttäjälle avautuu tyhjän lomakkeen sijaan komponentin vanhoilla tiedoilla esitäytetty lomake.

7.2.5 Komponentin poisto

Tällä sivulla käyttäjälle kerrotaan, että komponentin poistaminen ei tarkoita tietojen hävittämistä, vaan ainoastaan komponentin merkitsemistä poistetuksi (tila → poistettu). Tämän jälkeen komponentti ei ole oletusarvoisesti mukana haku- tai selailu-näkymissä.

7.2.6 Viimeisimmät yleiskomponenttien päivitykset

Tällä sivulla näytetään otanta viimeisimmistä yleisiä komponentteja koskevista muutoksista. Näytettävien muutosten määrä on rajattu siten, että taulukko mahtuu yhdelle sivulle.

7.2.7 Komponentin tiedot

Tällä sivulla esitetään kaikki komponentista kuvatut tiedot helposti tulostettavassa muodossa. Sivulta on myös linkit kuhunkin liitteeseen sekä Javadoc-dokumentaatioon.

Komponentin versioista ja muutoshistoriasta esitetään oletusarvoisesti vain viisi (5) uusinta merkintää, mutta halutessaan taulukot saa laajennettua näyttämään kaikki versiot ja muutokset.

Komponentin tiedot -sivulla on mahdollisuus päivittää kunkin version kuvausta. Tämä on poikkeama selainkäyttöliittymän päivitystoimintojen yleisestä linjasta, sillä kaikki muut päivitykset tehdään lisäys-, päivitys- tai poisto-sivujen kautta. Versioita ei näytetä komponentin päivitys -sivulla, sillä sivu täyttää jo nykyisellään normaalikokoisen selainikkunan. Ja jotta versioiden päivittämistä varten ei tarvitsisi luoda omaa sivuaan, päivitystoiminnallisuus toteutettiin komponentin tiedot -sivulle.

Poistetuksi merkittyjen komponttien kohdalla on erikseen korostettu, että komponentti on poistettu. Sen on tarkoitus ilmaista käyttäjälle, että hänen olisi syytä etsiä muita komponentteja, sillä tätä komponenttia ei luultavasti ole lupa käyttää.

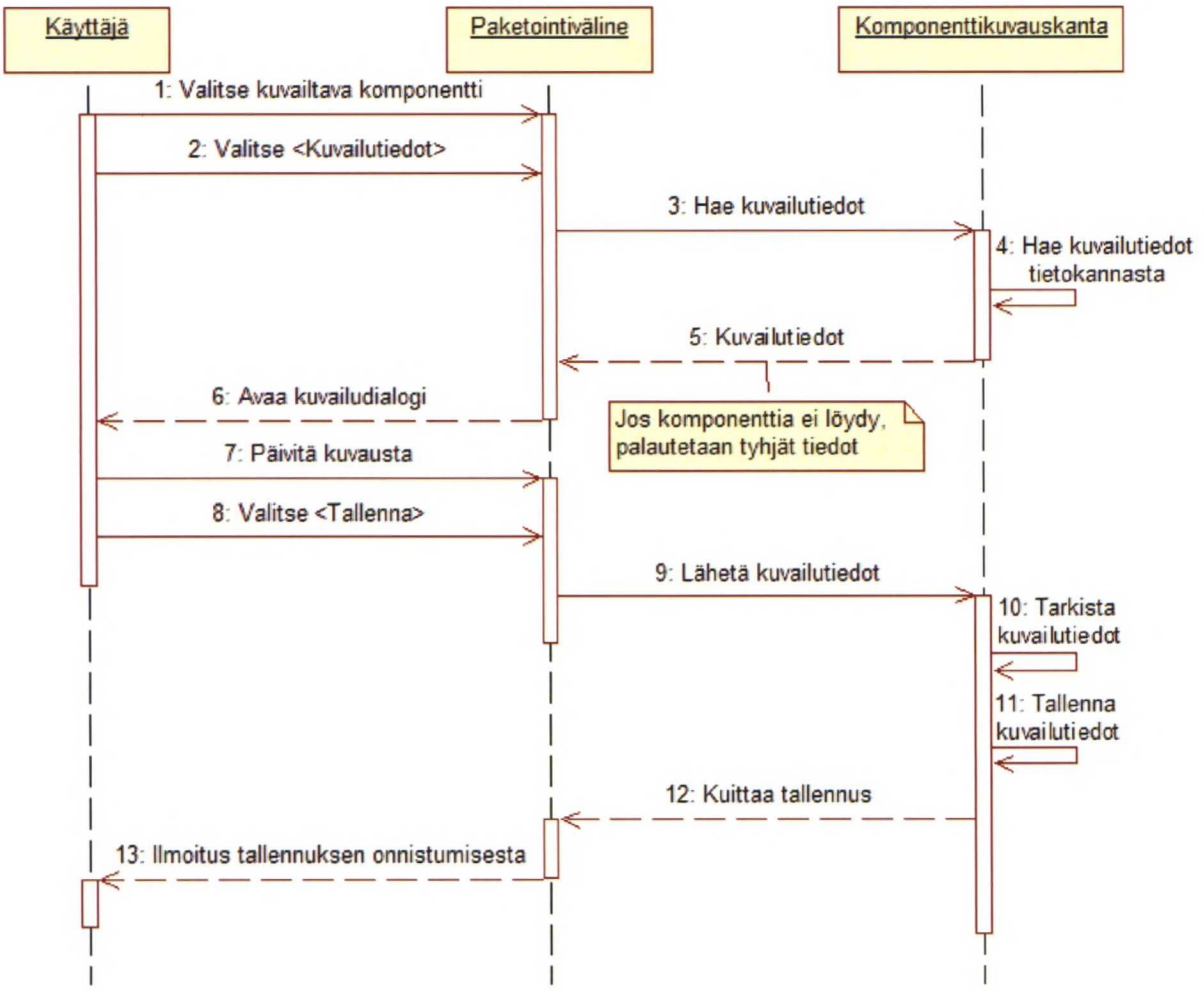
7.3 Liittymät

7.3.1 Paketointiväline

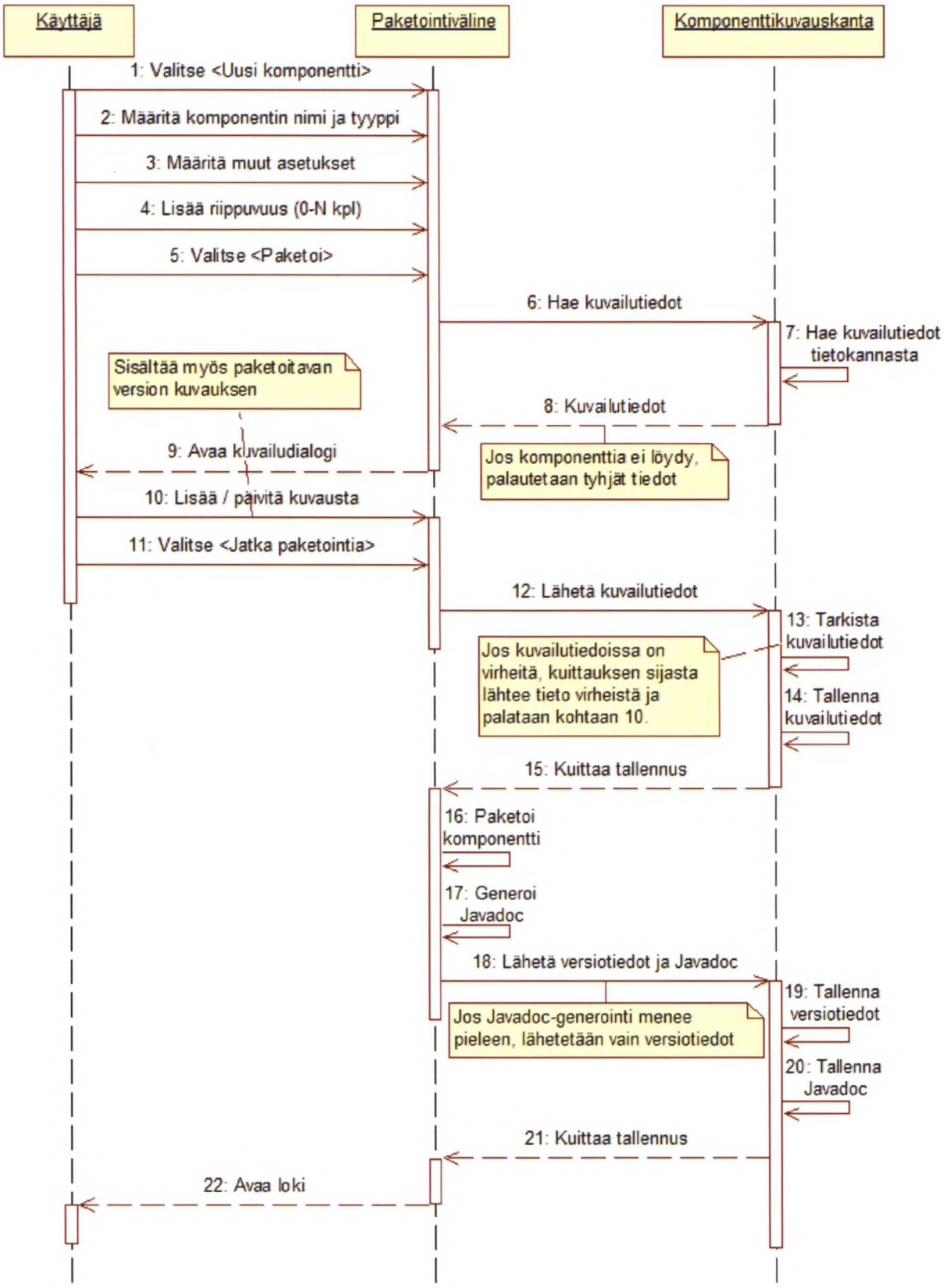
Paketointivälineellä on mahdollista paketoida uusia komponentteja ja olemassa olevien komponenttien uusia versiota sekä päivittää komponentin kuvausta komponenttikuvauskannassa.

Kuvailutietoja ei voi päivittää täysin samassa laajuudessa kuin selainkäyttöliittymän kautta. Liitetiedostojen käsittely ja versioon liitetyn kuvauksen päivittäminen on rajattu yksinomaan selainkäyttöliittymään. Versiokuvaus voi olla erilainen komponenttikuvauskannassa ja versionhallintajärjestelmässä, koska nämä on tallennettu eri paikkaan eikä komponenttikuvauskantaan tehty muutos ulotu versionhallintajärjestelmään.

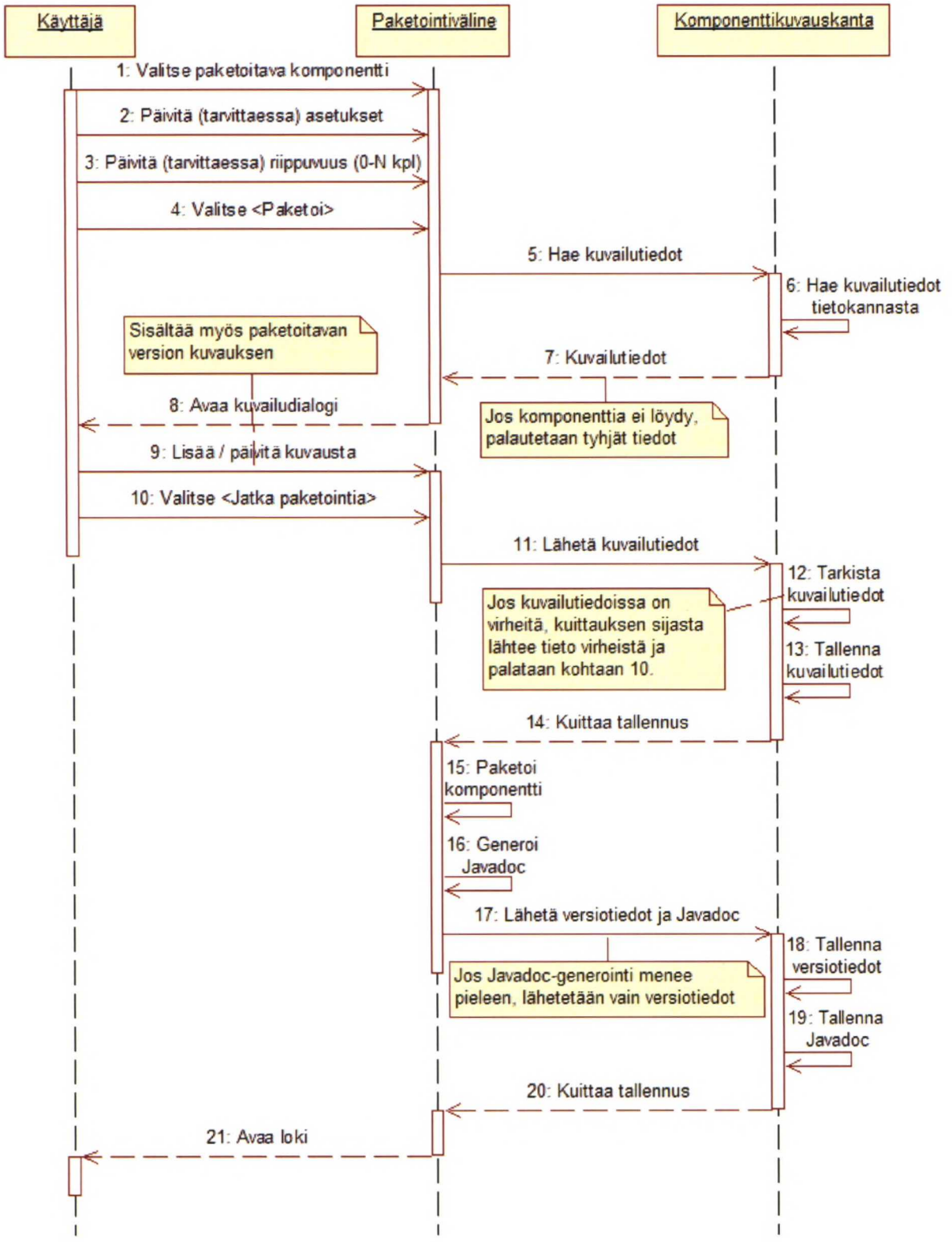
Paketointiväline on integroitu löyhästi komponenttikuvauskantaan eli paketointivälineen operatiiviset toimenpiteet eivät häiriinny, vaikka komponenttikuvauskannan palvelut eivät olisi käytettävissä toimenpiteen aikana. Seuraavien sivujen kuvissa [Kuva 9, Kuva 10, Kuva 11,] on esitelty edellä mainittujen kolmen toiminnon (ensimmäinen paketointi, paketointi ja kuvailutietojen päivitys) sekvenssikaaviot. Kaavioissa ei ole yksinkertaistamisen vuoksi kuvattu poikkeustilanteita eli kyseessä on vain onnistuneiden tapausten kulku (happy path).



Kuva 9 – Komponentin kuvailutietojen päivitys



Kuva 10 – Uuden komponentin paketointi



Kuva 11 – Komponentin paketointi

7.3.2 Versionhallinta

Komponenttikuvauskanta ei ole suorassa kanssakäymisessä versionhallintajärjestelmän kanssa. On kuitenkin toteutettu eräajo, jonka avulla voidaan siirtää aika ajoin versionhallinnasta löytyvät komponenttien versiotiedot komponenttikuvauskantaan. Näiden välille saattaa syntyä eroavaisuuksia, jos komponenttikuvauskanta ei ole saatavilla paketoinnin aikana. Eräajo ei tee muutoksia versionhallinnan tietoihin. Lisäksi se siirtää komponenttikuvauskantaan ainoastaan sieltä puuttuvien versioiden tiedot.

7.3.3 Konfiguraation hallinta -väline

Nykyisessä komponenttikuvauskannan toteutuksessa ei ole minkäänlaista integraatiota konfiguraation hallinta -välineeseen. Tilanne saattaa tulevaisuudessa muuttua, kun konfiguraation hallinta -väline on otettu laajalti käyttöön kohdeyrityksessä. Varsinkin komponenttien vastuuhenkilöille olisi hyödyllistä nähdä komponenttiensa todellista käyttöä koskevia tietoja ja tilastoja, mutta tämä ei välttämättä vaadi integraatiota komponenttikuvauskantaan ellei se tarjoa jotain lisäarvoa verrattuna siihen, että tietojen hankkimiseen käytettäisiin pelkkää konfiguraation hallinta -välinettä.

7.4 Teknologiat

7.4.1 Käyttäjän työaseman vaatimukset

Selainkäyttöliittymän käyttämiseksi työasemassa pitää olla asennettuna komponenttikuvauskannan kanssa yhteensopiva WWW-selain (esim. Microsoft Internet Explorer tai Mozilla Firefox). Paketointiväline, Microsoft Visual SourceSafe ja konfiguraation hallinta -väline vaativat käyttöjärjestelmäkseen Microsoft Windowsin. Käyttäjän tunnistamisessa hyödynnetään Microsoft Windows XP -käyttöjärjestelmän sisäänkirjautumistietoja. Työasemalla on oltava kohdeyrityksen paikallisverkko- tai etäyhteys.

7.4.2 Selainsovellus

Selainsovellus on toteutettu Apache Struts -MVC-kehyksen [Struts] päälle. JSP-käyttöliittymässä on hyödynnetty useita vakiintuneita Taglib-kirjastoja.

Tapahtumanhallinta on toteutettu EJB-teknologian avulla hyödyntäen Container Managed Bean-teknologiaa, jossa Java EE-palvelinohjelmisto pitää huolen tapahtumaketjujen eheästä suorittamisesta. Lisäksi toteutuksessa on hallittu rinnakkaisuutta oman tallennusaikaleima -toteutuksen avulla. Tietojen haun ja tallennuksen aikana ei sallita ko. aikaleiman vaihtumistam, sillä jos aikaleima ei täsmää, joku toinen taho on muokannut komponentin tietoja haku- ja tallennustapahtumien välisenä aikana.

Toteutuksessa on hyödynnetty useita Java EE -suunnittelumalleja (Java EE Design Patterns): Business Delegate, Business Interface, Session Facade, Service Locator, Data Access Object, Transfer Object, View Helper jne.

Kuvailutietojen haut ja tallennukset tehdään tietokantaan tallennettujen proseduurien (Stored Procedure) avulla. Lisäksi Javadoc-haun toteutuksessa hyödynnetään vapaan tekstin indeksoinnin ja haun tarjoavaa ohjelmistoa erillisten haku- ja tallennuskutsurajapintojen (Java API) kautta.

Selainsovellus on toteutettu itsenäisenä moduulina, joten se voidaan asentaa sellaisenaan eri palvelimelle kuin komponenttikuvauskannan haku- ja tallennuspalvelut tarjoava komponentti (EJB) on asennettu. Selainsovelluksen suoritusympäristönä voidaan käyttää mm. Portaalikehystä (esim. Bea WebLogic Portal [BEAWP]).

7.4.3 Liittymät

7.4.3.1 Paketointiväline

Paketointiväline on toteutettu Visual Basic -kielellä. Paketoinnissa hyödynnetään Apache Ant [Ant] -työkalua. Komponenttien operatiiviset tiedot (lähdekoodin sijainti, riippuvuudet jne.) tallennetaan versionhallintajärjestelmään. Paketointiväline keskustelee komponenttikuvauskannan kanssa HTTP-protokollan avulla. Komponenttikuvauskannan päässä kuuntelee Java Servlet, joka käyttää hyödykseen komponenttikuvauskannan palvelut tarjoavaa EJB-komponenttia.

Paketointiväline lähettää komponenttien kuvailutiedot, paketoidun version tiedot sekä ZIP-tiedostoksi pakatun Javadoc-dokumentaation multipart/form-data -tyyppisen HTTP-lomakkeen avulla komponenttikuvauskannan Servletille. Komponenttikuvauskanta vastaa näihin pyyntöihin aina XML-instanssilla, joka haun tapauksessa sisältää haettavan komponentin kuvailutiedot ja paketoinnin tai kuvailutietopäivityksen yhteydessä kuittauksen (OK, käyttäjän virhe, järjestelmävirhe jne.).

7.4.3.2 Versionhallintajärjestelmä

Kohdeyrityksessä käytetään Microsoft Visual SourceSafe [VSS] – versionhallintaohjelmistoa, joka ei ole asiakas-palvelinmallin mukainen vaan irrallinen työasematoteutus. Ainoastaan versionhallintajärjestelmän tietokanta on keskitetyssä paikassa varmistetulla verkkolevyllä. Paketointiväline hyödyntää integraatiossa työasemien Visual SourceSafe -ohjelmistoasennuksia.

7.4.3.3 Konfiguraation hallinta -väline

Konfiguraation hallinta -väline on toteutettu myös Visual Basic -kielellä. Se käyttää paketointivälineen tapaan versionhallintajärjestelmää tietovarastonaan. Sen toiminnan kannalta on myös oleellista, että se pääsee käsiksi paketointivälineellä kuvattuihin komponenttien välisiin riippuvuustietoihin mm. automatisoitujen EJB-client -päivitysten toteuttamiseksi.

8. TOTEUTUKSEN ARVIOINTI

8.1 Arviointi

Käytännön kokemuksia toteutuksen onnistumisesta ei ehditty keräämään ennen tutkimuksen valmistumista, sillä komponenttikuvauskanta otetaan käyttöön vasta vuoden loppupuolella. Arviointi perustuukin lähinnä tutkimuksen aikana kerätyn teoreettisen pohjan ja toteutuksen väliseen vertailuun. Arvioinnissa pitää huomioida myös reaalimaailman rajalliset toteutusresurssit.

8.2 Taustaa

Komponenttikuvauskannan toiminnallisuus sekä komponenteista kuvattavat tiedot ja arvojoukkojen vaihtoehdot korreloivat yksinomaan kohdeyrityksen tarpeita. Nämä tarpeet kartoitettiin komponenttikuvauskanta kehittäjille -projektin alkupuolella haastatteluiden perusteella. Myöhemmin näitä tarpeita tarkennettiin auditointitilaisuuksien avulla, joissa pyydettiin lopullisen tuotteen käyttäjiä arvioimaan kunkin kuvailutiedon ja toiminnallisuuden tarpeellisuutta ja sisältöä.

8.3 Hyödyllisimmät ominaisuudet

Tärkein komponenttikuvauskannan saavutus on keskitetty kanavan tarjoaminen kaikkien kohdeyritysestä löytyvien komponenttien tarkasteluun.

Kaikkein tärkeimmäksi kuvailutiedoksi uskotaan muodostuvan Javadoc-rajapintadokumentaatio. Lähes kaikista komponenteista saatavilla olevan Javadoc-dokumentaation avulla kehittäjä voi tutkia komponenttien rajapintaa ja toiminnallisuutta tarkalla tasolla.

Tärkeänä koetaan myös versiokohtaisen tiedon esittäminen keskitetyssä paikassa, sillä käytännön kokemus on osoittanut komponentin elämänkaareen mahtuvan monta haasteellista vaihetta, jotka usein liittyvät versioiden väliseen yhteensopivuuteen sekä muutosten koordinointiin ja muutoksista tiedottamiseen. Komponenttikuvauskanta

keskitettynä tiedotuskanavana auttanee jatkossa komponenttien vastuuhenkilöitä näiden haasteiden voittamisessa.

Liitetiedostojen keskitetty ja varmennettu tallennuspaikka tuo myös lisäarvoa parantamalla liitteiden saatavuutta ja turvaamalla niiden olemassa olon. Nykyisin näitä dokumentteja säilytetään hyvin sekalaisissa paikoissa, joista niitä on hankala saada käsiinsä ja joiden kautta ne voivat lisäksi hukkua. Käyttäjä saattaa myös saada käsiinsä epäkurantteja versioita.

Ei pidä myöskään vähätellä käyttöesimerkkien merkitystä. Varsinkin laajempien palvelukomponenttien lähdekoodiin upotetut (Javadoc) käyttöesimerkit saattavat olla vaikeasti löydettävissä. Jos karkean tason käyttöesimerkit on lisäksi kuvattu ohjeiden yhteydessä, on kunkin käyttäjän helpompi ja nopeampi sisäistää ko. palvelun käyttötavat.

8.4 Epäilyksen kohteet ja tulevaisuuden haasteet

Eniten voidaan epäillä komponentin tila -tiedon tarpeellisuutta ja luotettavuutta. Tila -valinta on täysin komponentin kehittäjien omalla vastuulla eli se ei päivity mitenkään automaattisesti esim. konfiguraation hallinta -välineen siirtopyyntöjen toimesta. Tila on määritelty pakolliseksi tiedoksi, mutta se ei takaa tiedon ajantasaisuutta ja oikeellisuutta. Esim. komponentin tilaksi saattaa jäädä "kehityksessä", vaikka komponentti olisi todellisuudessa jo tuotantokäytössä.

Myös luokittelu-valintoihin uskotaan tulevaisuudessa kohdistuvan muutospaineita. Luokittelu-vaihtoehtojen sekä arvojoukkojen määritteleminen oli todella hankalaa. Tulevilla loppukäyttäjillä oli tästä monia eriäviä mielipiteitä ja lopullinen malli jäikin lähes yksinomaan toteuttajien harteille. Luokittelua ei ole kovin helppo muuttaa. Toteutus haluttiin tehdä mahdollisimman yksinkertaisesti ja nopeasti, mikä johti siihen, että muutos vaatii kohtuullisesti ohjelmointityötä ja testaamista.

Jos myöhemmin halutaan lisätä tai poistaa kuvailutietoja, se vaikuttaa kaikkeen toteutukseen: selainsovellukseen, palveluihin, paketointivälineeseen ja näiden väliseen liikenteeseen. Tältä työltä tuskin olisi vältytty, vaikka toteutuksessa olisi hyödynnetty metatietokuvaustuotetta, jossa kuvattavat tiedot olisi määritelty jonkin

metatietosanaston avulla, sillä joka tapauksessa muutos pitäisi toteuttaa paketointivälineen ja komponenttikuvauskannan väliseen integraatioon.

Komponenttikuvauskanta tuskin valmistuu yhdessäkään yrityksessä kerralla lopulliseen muotoonsa vaan jatkuvasti muuttuva sovelluskehitysala aiheuttaa muutospaineita myös komponenttikuvauskannalle. Siks myös kohdeyrityksessä komponenttikuvauskanta on otettava jatkuvan ylläpidon ja jatkosuunnittelun kohteeksi. Kaikki välineeseen liittyvä palaute on syytä käsitellä ja tallentaa huolella, koska se on kaikkein tärkein lähde muutoksia suunnitellessa. Palautettahan tulee nimenomaan loppukäyttäjiltä, joiden tarpeisiin välineellä on pääosin yritetty vastata.

9. YHTEENVETO

Tutkimuksen olennaisin anti on komponentteihin liittyvien käsitteiden määritteleminen ja rajaus, komponenttien uudelleenkäyttöön ja hallinnointiin sisältyvien seikkojen pohdinta, hallintavälineiden käytöllä parantuvan kehittämisen tehokkuuden arviointi sekä havainnot hallinnointivälineiden välisestä työnjaosta ja integroinneista.

Komponentti-käsite rajattiin käytännön näkökulmasta, sillä sen määriteltiin olevan Java-teknologian tavalla paketoitu käännöstiedostojen kooste. Lisäksi komponentin laskettiin sisältävän sen toimintaan kiinteästi liittyvät resurssi- ja konfiguraatiotiedostot. Komponentti-käsitteen selkeästä määrittelystä on hyötyä yrityksen viestinnässä. Rajapinta- ja komponenttikäsitteiden tiukka yhteen nitominen on myös tärkeä havainto. Rajapintapainotteinen komponenttien suunnittelu tehostaa komponenttien uudelleenkäyttöä ja pienentää niiden ylläpitokustannuksia.

Komponenttikuvauskantaa ja sen yhteydessä käytettyä terminologiaa voidaan hyödyntää yritysten viestinnän tehostamisessa. Komponenttien kuvaaminen mahdollistaa myös yhteisten standardien ja linjausten muodostamisen yrityksissä. Myös tiedon määrä ja sen kulku nopeutuu keskitetyn kanavan ansiosta.

Koska tieto komponenteista leviää paremmin, myös komponenttien uudelleenkäyttö lisääntyy. Tutkimuksen aikana kuitenkin havaittiin, että komponenttikuvauskanta ei yksinään riitä ratkaisemaan komponenttien uudelleenkäyttöön liittyviä ongelmia. Uudelleenkäyttö vaatii yritykseltä paljon muutakin: johdon sitoutumista, investointeja resurssien muodossa, infrastruktuurin kehittämistä uudelleenkäyttöä tukevaksi ja standardisointia edistäväksi, komponenttien vastuuttamista ja omistussuhteiden muodostamista, kehittäjien koulutusta jne. Tutkimuksessa myös todettiin uudelleenkäyttöön perustuvan sovelluskehitysmallin (CBSD) olevan nykypäivänä vielä utopiaa.

Komponenttien keskitetty hallinnointi ja yhteinen tiedotuskanava nopeuttavat ajantasaisen tiedon löytymistä, mikä lisää kehittämisen tehokkuutta. Myös toisteisen tiedon vähentäminen tehostaa kehittämistä. Toisteisuutta vältetään mm. hallintavälineiden integroinneilla. Tehokkuuttaa voidaan lisätä mm. mahdollistamalla

komponenttien kuvaaminen paketointivälineellä ja välittämällä paketoitujen versioiden tiedot automaattisesti komponenttikuvauskantaan. Sovelluskehityksen hallintavälineiden käytöllä pienennetään myös inhimillisten virheiden riskiä, mikä parantaa komponenttien laatua ja nostaa kehittämisen tehokkuutta.

Tutkimuksen pohjalta päädyttiin komponenttikuvauskannan omaan toteutusmalliin, koska markkinoilla olevien tuotteiden kartoitus tuotti erittäin laihat tulokset. Toteutusmallin tärkeimmät osatekijät ovat kattava kooste Java-komponenteista kuvattavista tiedoista sekä komponenttikuvauskannan ja sen liittymien toiminnallisuuden määrittely.

10. JATKOTUTKIMUSMAHDOLLISUUDET

Erityisesti kaksi tutkimuksen aikana esille noussutta ideaa antavat aihetta jatkotutkimuksiin: interaktiivinen kehitysympäristö ja muutosten vaikutusalueen kartoitus (Impact Analysis tai Change Management).

Interaktiivinen kehitysympäristö [Ye00] saattaisi tuoda komponenttikuvauskannan huomattavasti lähemmäksi sovelluskehittäjän arkea. Interaktiivinen tiedonvälitys poistaisi tietosaarekkeita ja lisäisi siten komponenttien tunnettuvuutta, mikä taas lisäisi uudelleenkäyttöä. Tähän kehitysvälineiden ja komponenttikuvauskannan integrointiin liittyy kuitenkin merkittäviä haasteita, joihin jatkotutkimusten tulisi pureutua.

Sovelluskehityksen muutoshallinnan haastavuus kasvaa yhdessä uudelleenkäytön kanssa. Uudelleenkäyttö on nykyisen trendin mukainen kehityssuunta, joten muutosten vaikutusalueen kartoittamiseen tulisi panostaa huomattavasti aiempaa enemmän. Sovellusten elinkaaren pitkäkestoisin vaihe on ehdottomasti sen tuotantokäyttö, mikä häiriintyy, jos komponentteihin tehtyjä muutoksia ei hallinnoida huolella. Tässä erityiset Impact Analysis -välineet olisivat suureksi avuksi. Muutosten vaikutusaleen kartoitus on tällä hetkellä erityisen kiinnostuksen kohteena sekä akateemisessa tutkimuksessa että teollisuudessa, joten tältä tutkimusalueelta on odotettavissa konkreettisia tuloksia tulevaisuudessa.

LÄHDELUETTELO

[Ant] Apache Ant, Viitattu 21.6.2006, http://ant.apache.org/.

[AntEJB] Ant EJB Tasks, Viitattu 21.6.2006, http://ant.apache.org/manual/OptionalTasks/ejb.html.

[Struts] Apache Struts -framework, Apache Software Foundation, Viitattu 31.8.2006, http://struts.apache.org/.

[BEAWP] BEA WebLogic Portal, Viitattu 10.8.2006, http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/portal/.

[Beh98] Anita Behle, An Internet-based Information System for Cooperative Software Reuse, Proceedings of the Fifth International Conference on Software Reuse, pp. 236-245, June 2-5, 1998, Victoria, British Columbia, Canada.

[Boe99] Boehm, B., Managing software productivity and reuse, Computer, vol.32, no.9, pp.111-113, Sep 1999.

[Boo87] Grady Booch, Software Components with Ada: Structures, Tools and Subsystems, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA, 1987.

[Bor95] J. Borstler, Feature-oriented Classification for Software Reuse, Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering (SEKE'95), pp.204-211, June, 1995, Rockville, ML, Knowledge Systems Institute, Skokie, IL, USA.

[Bro87] Brooks, F. P. Jr., No Silver Bullet: Essence and Accidents of Software Engineerig, Computer, vol. 20, no. 4, pp. 10-19, April 1987.

[BST] Borland StarTeam, Viitattu 31.8.2006, http://www.borland.com/us/products/starteam/.

[Can06] G. Canfora and L. Cerulo, Jimpa: An Eclipse Plug-in for Impact Analysis, Proc. of the 10th Conference on Software Maintenance and Reengineering – CSMR 2006, Bari, Italy, IEEE Computer Society Press, pp. 341-342

[Car94] Card, D.; Comer, E., Why do so many reuse programs fail?, Software, IEEE , vol.11, no.5, pp.114-115, Sep 1994.

[Cer06] Luigi Cerulo, On the Use of Process Trails to Understand Software Development, Ph.D. Thesis, Università Degli Studi Del Sannio, 82100 Benevento, Italy, 30.6.2006.

[Cle95] Clements, Paul C., From Subroutines to Subsystems: Component-Based Software Development, 3-6. Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Crn02] Crnkovic, I., Hnich, B., Jonsson, T., and Kiziltan, Z. 2002. Specification, implementation, and deployment of components. Commun. ACM 45, 10 (Oct. 2002), 35-40.

[CVS] Concurrent Versions System, Viitattu 21.6.2006, http://www.nongnu.org/cvs/.

[Fav98] Favaro, J. M., Favaro, K. R., and Favaro, P. F. 1998. Value based software reuse investment. Ann. Softw. Eng. 5 (Jan. 1998), 5-52.

[Fra96] Frakes, W.B.; Fox, C.J., Quality improvement using a software reuse failure modes model, Software Engineering, IEEE Transactions, vol.22, no.4, pp.274-279, Apr 1996.

[Gil04] Gill, N. S., Grover, P. S., Few important considerations for deriving interface complexity metric for component-based systems. SIGSOFT Softw. Eng. Notes 29, 2 (Mar. 2004), 4-4.

[Gil06] Gill, N. S., Importance of software component characterization for better software reusability. SIGSOFT Softw. Eng. Notes 31, 1 (Jan. 2006), 1-3.

[Gri99] Martin L. Griss, Architecting for Large-Scale Systematic Component Reuse, In Proceedings of the International Conference of Software Engineering'99, 1999, Los Angeles, CA, USA.

[Guo00] Guo, J.; Luqi, A survey of software reuse repositories, Engineering of Computer Based Systems, 2000. (ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshop, pp.92-100, 2000

[Hai97] Capt Gary Haines, David Carney, John Foreman, Component-Based Software Development / COTS Integration, 1997, Viitattu 29.6.2006, http://www.sei.cmu.edu/str/descriptions/cbsd.html.

[Hen96] Henninger, S., Supporting the construction and evolution of component repositories, Software Engineering, 1996., Proceedings of the 18th International Conference, pp. 279-288, 25-29, Mar, 1996.

[Hol01] Hollanti, Marko, Uudelleenkäytettävien ohjelmistokomponenttien rajapintojen dokumentointi, Tietojenkäsittelytieteiden laitos, Jyväskyän Yliopisto, 2001.

[Hop00] Hopkins, J., Component primer. Commun. ACM 43, 10 (Oct. 2000), 27-30.

[IRCCM] IBM Rational ClearCase Change Management Solution, Viitattu 31.8.2006, http://www-306.ibm.com/software/awdtools/changemgmt/.

[iPlaCL] iPlanet Application Server Developer's Guide: Appendix B Runtime Considerations, Viitattu 21.6.2006, http://docs.sun.com/source/816-5786-10/jpgapxre.htm.

[JWS] Java Technology and Web Services, Sun Microsystems, Inc., Viitattu 10.8.2006, http://java.sun.com/webservices/.

[Mey02] Anthony Meyer, Software Reuse: Why You Need a Well Managed Repository, Pearson Education, Inc., Informit, 800 East 96th Street Indianapolis, Indiana 46240, Apr 5, 2002, Viitattu 20.7.2006, http://www.informit.com/articles/article.asp?p=26227&rl=1.

[OSLic] Open Source Initiative, The Approved Licenses, Viitattu 4.7.2006, http://www.opensource.org/licenses/.

[Par03] Rebecca Parsons, "Components and the World of Chaos", IEEE Software, vol. 20, no. 3, pp. 83-85, May/Jun, 2003.

[Pri91] R. Prieto-Diaz, Implementing Faceted Classification for Software Reuse, pp. 8897, Communications of the ACM 34(5), ACM, New York, May, 1991.

[RDF] Resource Description Framework (RDF) Primer, W3C, Viitattu 23.8.2006, http://www.w3.org/TR/rdf-primer/.

[Ren04] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara Ryder, and Ophelia Chesley, Chianti: A Tool for Change Impact Analysis of Java Programs, In Proceedings of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04), (Vancouver, Canada, October 2004), pp. 432-448.

[Ren05] Xiaoxia Ren, Barbara Ryder, Maximilian Storzer, and Frank Tip, Chianti: A Change Impact Analysis Tool for Java Programs, In Proceedings of the 27th International Conference on Software Engineering (ICSE'05) (Research demonstrations Session), (St. Louis, MO, USA, May 2005), pp. 664-665.

[Ros95] Rosenbaum, S., B. DuCastel, Managing Software Reuse--An Experience Report, Proc. 17th ICSE, 105-111, 1995, Seattle, Washington.

[Sam97] J. Sametinger, Software Engineering with Reusable Components, Springer-Verlag, ISBN 3-540-62695-6, 1997.

[SHW98] Robert C. Seacord, Scott A. Hissam and Kurt C. Wallnau, Agora: A Search Engine for Software Components, Technical Report, CMU/SEI-98-TR-011, ESC-TR-98-011, August 1998.

[SIR] Profium SIR – Semantic Information Router – metatietokuvauskantatuote, Profium Ltd., Viitattu 23.8.2006, http://www.profium.com/products/sir.html.

[Subv] Subversion, Viitattu 21.6.2006, http://subversion.tigris.org/.

[Tör05] Törngren, M.; DeJiu Chen; Crnkovic, I., Component-based vs. model-based development: a comparison in the context of vehicular embedded systems, Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference, pp. 432- 440, 30 Aug.-3 Sept. 2005

[Ven04] Venäläinen Helena, Uudelleenkäytön nykytila ja edistämismahdollisuudet uudelleenkäyttökokemusten pohjalta, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 19.5.2004.

[Vit03] Vitharana, P. 2003. Risks and challenges of component-based software development. Commun. ACM 46, 8 (Aug. 2003), 67-72.

[VSS] Microsoft Visual SourceSafe, Viitattu 21.6.2006, http://msdn.microsoft.com/ssafe/.

[Wal92] Walton, P., The Management of Reuse, Software Reuse and Reverse Engineering in Practice, P.A.V. Hall, ed., Chapman & Hall, London, 1992, pp. 505-520.

[WebSp] Tim deBoer, Gary Karasiuk, J2EE Class Loading Demystified, Viitattu 4.7.2006, http://www-128.ibm.com/developerworks/websphere/library/techarticles/0112_deboer/deboer.html.

[Wiki] Wiki-käsitteen määritelmä, Suomalainen Wikipedia, Viitattu 10.8.2006, http://fi.wikipedia.org/wiki/Wiki.

[WkVer] Kattava lista erilaisista versionhallintaohjelmistoista, Englanninkielinen Wikipedia, Viitattu 21.6.2006, http://en.wikipedia.org/wiki/List_of_revision_control_software.

[WLSCL] WebLogic Server Application Classloading, Viitattu 21.6.2006, http://e-docs.bea.com/wls/docs81/programming/classloading.html.

[Yac99] Sherif Yacoub, Hany Ammar, Ali Mili, Characterizing a Software Component, 1999, CSEE Department, West Virginia University, Morgantown, WV 26506.

[Ye00] Yunwen Ye, Gerhard Fischer, Brent Reeves, Integrating active information delivery and reuse repository systems, Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering: twenty-first century applications, p.60-68, November 06-10, 2000, San Diego, California, United States.

[Zha01] Lu Zhang; Hong Mei; Hong Zhu, A configuration management system supporting component-based software development, Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International , vol., no.pp.25-30, 2001.